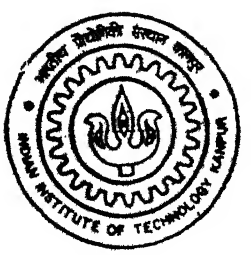
IMPLEMENTATION OF RELIABLE MULTICAST TRANSPORT PROTOCOL FOR REMOTE TUTOR

by
Alpna Dutta



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
March, 2000

IMPLEMENTATION OF RELIABLE MULTICAST TRANSPORT PROTOCOL FOR REMOTE TUTOR

A Thesis Submitted

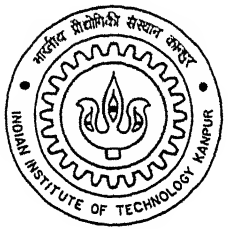
in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Alpna Dutta



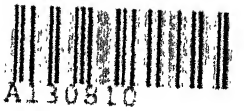
to the

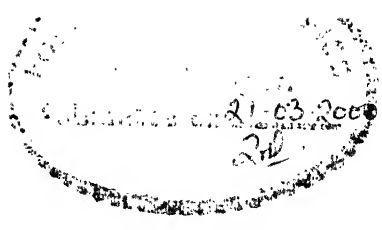
DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

INDIA

March 2000





CERTIFICATE

This is to certify that the thesis work entitled "**IMPLEMENTATION OF RELIABLE MULTICAST TRANSPORT PROTOCOL FOR REMOTE TUTOR**" by Alpna Dutta, Roll No. 9810404 has been carried out under my supervision and the same has not been submitted else where for any degree or diploma.

21/03/2000

Thesis Supervisor,
Dr. Y.N.Singh,
Assistant Professor,
Deptt. of Electrical Engg,
Indian institute Of Technology,
Kanpur-208016.

March 2000.

Abstract

In this thesis implementation of a reliable multicast transport protocol for *Remote Tutor* is reported. The *Remote Tutor* is a MS Windows based tele-seminaring tool, which is capable of creating a virtual classroom, comprising of a teacher and students. The members of the classroom could be geographically distributed according to the span of the network. In *Remote Tutor*, the student and teacher can interact with each other using graphics, video, text and annotations. The teacher can play a recorded lecture file, which consists of audio, video and the timestamped sequence of commands to create objects, open files etc. The transmission of these commands requires reliablity to avoid disturbances, which destroys the overall effect of classroom during an on going session. The reliablity requirement of such type of data is fulfilled by implementing a Tree-based Reliable Multicast Transport Protocol (TMTP) on top of the transport layer.

TMTP takes the advantage of the IP multicast for packet routing and delivery. However, for the purpose of scalable flow and error control, it dynamically organizes the participants into hierarchical control tree. The tree consisits of Sender, Domain Managers (DMs) and Group Members (GMs). A common source code is written for all three modules i.e., Sender, DM and GM, with corresponding flags to initialize them at the start of execution. The program has been tested for all the three modules, using a maximum of three routers in the network. The performance with such an environment has been found to be satisfactory.

Dedicated

to my

Grandparents and Parents

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my thesis supervisor Dr. Y. N. Singh for providing timely help and encouragement. I would also thank Mr. Vivek Mudgil for his guidance which enabled me in my endeavour to reach my goal.

I am also grateful to Mrs. Madhu Bajpai, Mrs. Neeru Chhabra, Akanksha, Seema, Manoj, Vineet and Kaushalji for their extended help during the course of my thesis. I would cherish the nice company of my batchmates Harish, Nitin, Gaurav, Praveen, Prabhat, Murthy and Bhuwanendra who maintained a benign work atmosphere at ERNET Lab. I also thank Mr. P. R. Sahu and Sudeep for their timely help.

I would not forget the unerasable moments spent with the John family, Daniel family, Manoharan family, Abraham family, Mrs. Ludmila Choudhary and Maymol Didi which made my stay at IITK a memorable one.

At this time I must not forget to mention the names of my friends Dhaval, Sharmila, Kusum and Manisha who were always besides me. I will always treasure those moments spent with them.

Last but not the least, I want to acknowledge the constant support and blessing which my parents gave me.

Alpna

Contents

...

List of Figures

List of Abbreviations

LMS : Lecture Manager and Store House.

URL : Uniform Resource Locator.

RTP : Real Time Protocol.

TMTP : Tree-Based Multicast Transport Protocol.

ACK : Acknowledgement packet.

NACK : Negative Acknowledgement packet.

MDP : Multicast Dissemination Protocol.

AFDP : Adaptive File Distribution Protocol.

MFTP : Multicast File Transfer Protocol.

RMTP : Reliable Multicast Transport Protocol.

DM : Domain Manager.

GM : Group Member.

TTL : Time-to-live value of a packet.

Chapter 1

INTRODUCTION

The objective of this work is to implement a reliable multicast protocol for *Remote Tutor* application. *Remote Tutor* is a MS Windows based tele-seminaring tool which is aimed at facilitating lecture/seminar delivery over Internet onto the paticipant's desktop. It supports interactive, real time seminar delivery over Internet. The *Remote Tutor* is capable of creating a virtual classroom, comprising of a teacher and students. The members of the classroom could be geographically distributed according to the span of the network. *Remote Tutor* allows the teacher to present lectures using text, viewgraphs (like the transparencies used with an overhead projector during a traditional seminar), graphics, audio and video files. The viewgraphs can be annotated by the teacher. Student can ask questions and make comments during the sessions using either a whiteboard wherein they can write or draw anything or by speaking up. This package is intended to be an application over high speed network for distance education. The network is assumed to support TCP/IP with multicast capability. The modular diagram of *Remote Tutor* application is shown in the Figure 1.1. It consists of three parts Tutor application, Student application and Lecture Manager Server. The functions of Tutor and Student application has been mentioned above.

Third component of *Remote Tutor* is Lecture Manager and Store House (LMS). It is a dedicated server performing the functions of the session management, database

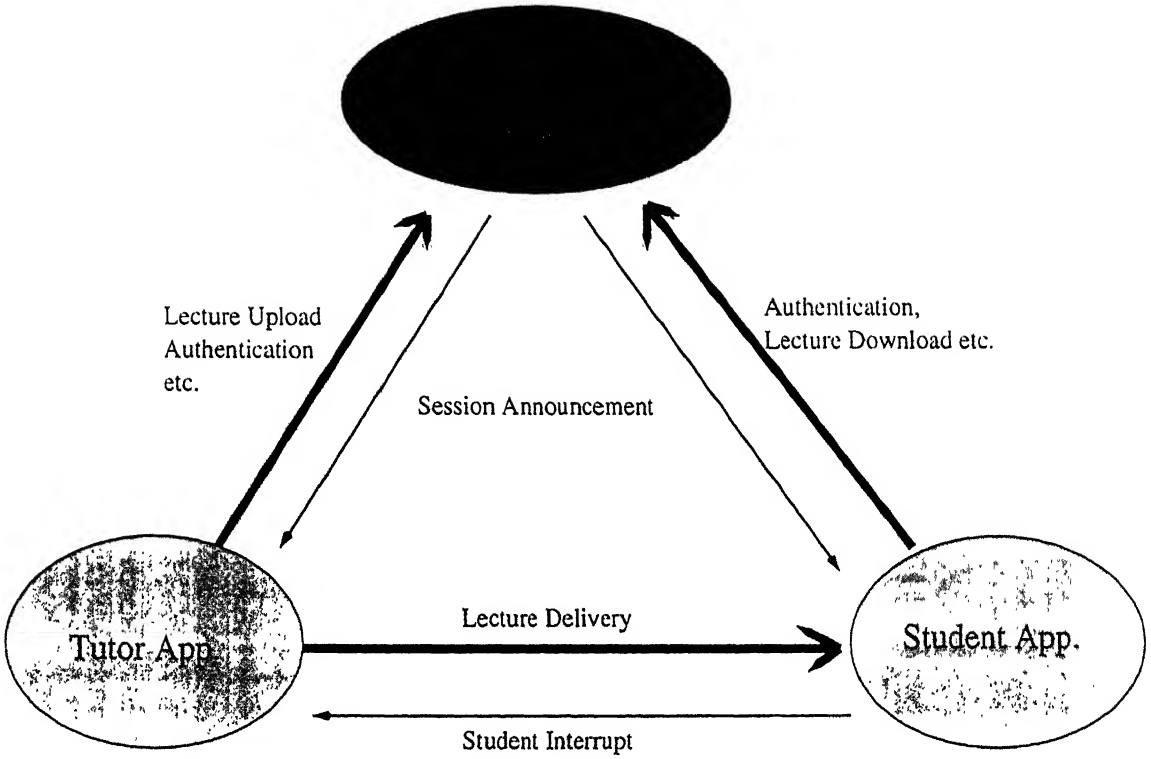


Figure 1.1: Modular Diagram of the *Remote Tutor* Application

management and authentication of student and tutor. As this application (LMS) is purely web based, student and teacher can access it by a known URL. Students and teachers have to be registered at LMS prior to attending or delivering a lecture. LMS database will keep information of registered student and teachers along with the course and lecture details including schedules and lecture materials. In essence, all the session management between student and teacher will be done by LMS.

1.1 Need of Reliable Multicast Transport Protocol in *Remote Tutor*

In *Remote Tutor*, the student and teacher can interact with each other using graphics, audio, video, text and annotations. The teacher can either play a recorded lecture or can deliver it online. The recorded lecture file consists of the audio, video and the time stamped sequence of commands to create objects, open files etc. The transmission of

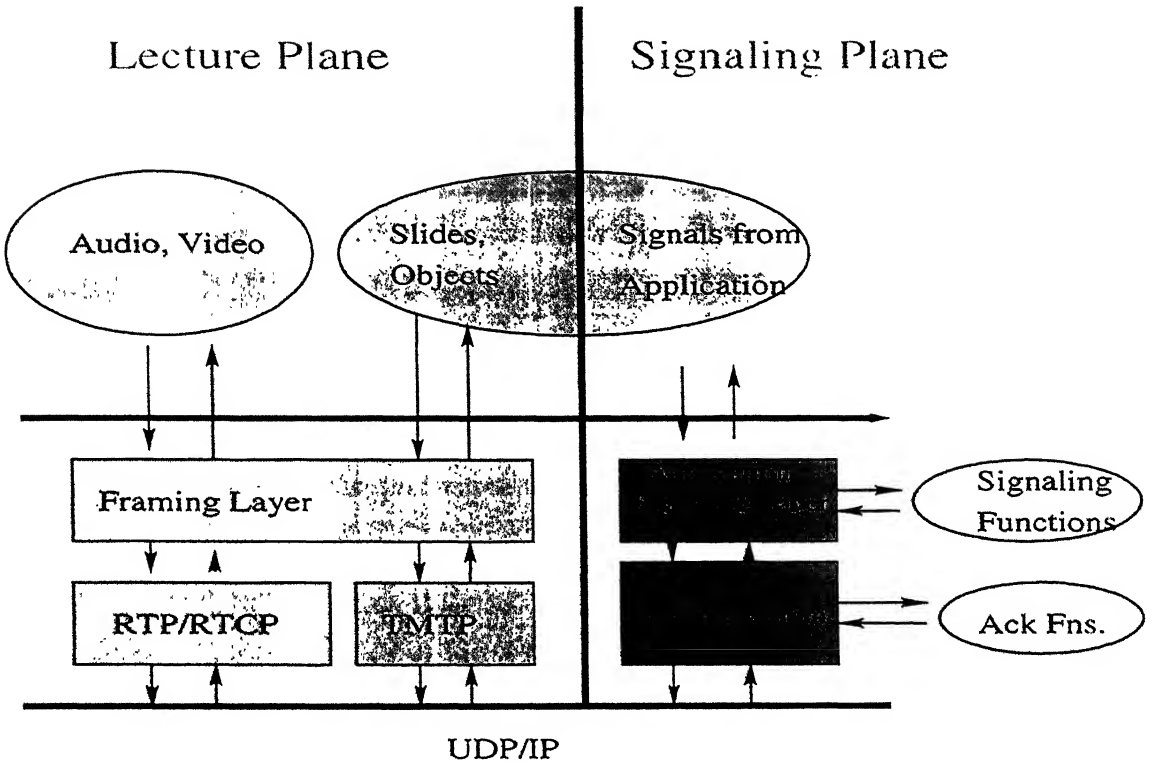


Figure 1.2: Network Interface of the *Remote Tutor* Application

these commands requires reliability to avoid disturbances, which destroys the overall effect of classroom during an ongoing session.

The transmission takes place on a multicast group address. The multicast traffic is handled at the transport layer using UDP (User Datagram Protocol), which is unreliable and connectionless. TCP (Transmission Control Protocol) is not feasible for multicast traffic, since TCP provides point-to-point connections. To provide reliability to the multicast transmission, a layer is added on top of transport layer. This is shown in the Figure 1.2.

In *Remote Tutor*, the application is divided primarily into two planes, signaling plane and lecture plane. While the signaling plane handles the application related signals between tutor, student and LMS, the lecture plane takes care of delivery and reception of lecture material. The lecture material constitues of audio/video streams requiring real-time service and sequence of commands requiring reliable delivery of data. This new layer is part of the *Remote Tutor* application itself. In *Remote Tutor*,

this is achieved by implementing two different protocols on the layer above the transport layer. One is for real time data, i.e. the data which has jitter as the prime concern, for example audio. Such real time data uses the the Real Time Protocol (RTP) for efficient transmission. The other type of data are the commands, which need the assurance of reaching the destination, to keep the smooth progress of the virtual classroom activity. The reliability requirement of such type of data is fulfilled by using a protocol called Tree-Based Multicast Transport Protocol (TMTP).

This thesis discusses the implementation of TMTP protocol. TMTP takes advantage of the IP multicast for packet routing and delivery. However, for the purpose of scalable flow and error control, it dynamically organizes the participants into hierarchical control tree. This control tree helps to distribute the functions of state management and error recovery among many members, thereby allowing protocol scalability to large number of receivers.

1.2 Scope of thesis

The first aim of the thesis has been to search and choose a Reliable multicast transport protocol matching the requirements of application in mind, from the various protocols available. After extensive literature survey, including search on the Internet, TMTP has been chosen for implementation. The TMTP protocol has been slightly modified depending on the requirements of the *Remote Tutor* application. Then, the TMTP protocol has been implemented and tested.

1.3 Organization of thesis

In Chapter 2, various available reliable multicast protocols have been discussed and the reason for the selection of TMTP protocol is also mentioned. The Chapter 3, gives the details of the TMTP protocol and also describes about the modifications done

in the TMTP protocol. In Chapter 4, the design and flow diagrams of the various modules of the TMTP are explained. The Chapter 5, gives the implementation details of the TMTP protocol. Finally, Chapter 6 contains the concluding remarks about the implementation.

Chapter 2

Various available reliable multicast transport protocols

In the recent times, the increase in popularity of groupware applications such as real-time conferencing tools and multipoint data distribution services have resulted in the development of reliable multicast transport protocols. Earlier multicast transport protocols were designed as a general solution to the group communication problem. Untill recently, it has been recognized that group communication application have a much wider range of requirements than the unicast applications.

In our application we require a TCP-like reliability. Since TCP cannot be used for multicasting and UDP cannot provide reliability, though it is capable of multicasting. Therefore, we need to design a multicast transport protocol which is specific to our requirement.

2.1 Criteria for protocol selection

As mentioned that the multicast protocols are designed for a specific purpose, there are some technical characteristics which can be used to distinguish them [1]. The multicast

transport protocols can be classified according to the following features.

2.1.1 Data propagation

Data propagation describes the mechanism used to propagate data (not control information) among participating sites.

2.1.2 Reliability mechanism

Traditional reliable unicast transport protocols, such as TCP, use ACKs to recover from packet loss. This approach to achieving reliability is often referred to as sender-initiated, since it is the responsibility of the sender to detect packet losses. In a multicast environment, as group sizes increase, the sender-initiated scheme may cause ACK implosion since each delivered packet triggers an acknowledgement from every receiver in the group.

Alternatively, in the receiver-initiated approach to reliability, receivers detect packet losses and request their retransmission by generating NACKs. Placing the responsibility of recovering from packet losses in the receiver helps alleviate the ACK implosion problem. The performance comparision study presented in [5] confirms that receiver-initiated multicast transport protocols deliver better performance than their sender-initiated counterparts.

2.1.3 Repair request and Retransmission

In reliable transport protocols, repair requests and retransmissions may be propagated point-to-point or to the whole group. In some protocols, a receiver unicasts a NACK to the sender to request retransmission of a lost packet. In response, the sender multicasts the repair to the whole group assuming that other receivers may have lost the same packet. Other protocols may go a step further, and use multicast to propagate NACKs

as well. The idea is to supress NACKs from other receivers that lost the same packet.

Some protocols propose the concept of local recovery. The goal is to try to recover from packet loss without having to go all the way to the source. Certain servers may be organized hierarchically and perform error recovery on behalf of the set of receivers for which they proxy.

2.1.4 Feedback control

In multicast transmission, if all the receivers respond to the source, then there is a feedback implosion at the source. To avoid feedback implosion, multicast protocols should provide some form of feedback control, i.e., a mechanism that restricts the amount of feedback generated by the multicast group.

The solution to the feedback implosion problem are classified as structure-based or timer-based. In the first classification, structure-based approaches rely on a designated site to process and filter feedback information. The second classification provides a more general definition of structure-based feedback control and refers to the solutions in which protocols organize multicast group members into some structure in order to filter the amount of feedback generated by the group.

Timer-based solutions rely on probabilistic feedback supression to avoid implosion at the source. Receivers in these protocols delay their retransmission requests for a random interval, uniformly distributed between the current time and the one-way trip time to the source. The goal is that group members closer to the source send their feedback sooner, supressing feedback from farther away members.

2.1.5 Flow and Congestion control

Controlling the packet rate in multicasting is complicated by the fact that the protocol must accommodate multiple receivers simultaneously. How this is performed can have

significant impact on the overall performance. For, example, if the sender always operates at the rate of the slowest receiver, then faster receivers sit idle while they wait for additional information. Hence, based on the feedback from the receivers, the source can dynamically adjust the transmission rate. This adjustment of transmission rate is done after regular time intervals.

2.1.6 Locus of control

Some protocols assign certain control functions to a central site, which handles error detection and retransmission. Others may use a control tree for error recovery and hence has a distributed control.

2.1.7 Ordering

Although ordering is not a requirement of reliable multicast, many of the protocols use it for order-critical applications such as file transfers and collaborative sessions.

2.1.8 Group management

Some multicast protocols do not require knowledge of group membership. Sources simply send data to a multicast group address and receivers can join to receive the data. This implicit group management model matches the way IP multicast handles group membership and has better scalability properties than explicit group management.

Other protocols may require full control over group membership. In this case, either membership is static throughout the session, or an explicit group membership protocol ensures reliable joins and leaves.

2.1.9 Target Application

Some multicast transport protocols address the requirements of delay-sensitive real-time services, such as multimedia conferencing tools. These applications can tolerate a certain degree of data loss, but are sensitive to packet delay variance. On the otherhand, traditional data distribution services, such as multipoint file transfer, are not delay-sensitive but require that objects be delivered in their entirety or the transfer fails.

2.1.10 Scalability

Since every multicast protocol must be able to communicate with a set of receivers, it is useful to have a rough estimate of how large the receiver set can be.

Most multicast transport protocols emphasize scalability as a fundamental design goal. Designing a multicast transport protocol that scales to large group sizes influences the design of several of the functions listed above. For instance, protocol that needs to scale should use receiver-based reliability through NACKs and employ some form of feedback control to avoid feedback implosion.

2.2 Comparision of the Reliable Multicast Protocols available

2.2.1 Multicast Dissemination Protocol (MDP)

The multicast dissemination protocol [6] provides a reliable multicast framework for file distribution. The MDP sender fragments the file into a sequence of MDP maximum data units (MDUs) which are multicast to the group using the UDP/IP multicast suite. MDP receivers assemble the received data units into the original file. MDP's recovery mechanism works in rounds: after transmitting a file, the source asks receivers for

retransmission requests. A receiver that has detected sequence number gaps schedules a retransmission request which gets multicast. The MDP source can also request ACKs from receivers, which respond with information about their current state. MDP allows sites to join and leave the multicast group.

2.2.2 Adaptive File Distribution Protocol (AFDP)

AFDP [7] also provides a reliable file distribution service on top of UDP. AFDP senders, or publishers send data to AFDP subscribers. A subscriber in a group is responsible for managing group membership, authorizing publishers, and determining the appropriate transmission mode to be used. While AFDP uses multicast as its preferred transmission mode, AFDP packets can also be transmitted using broadcast or unicast whenever multicast is not available. To achieve reliablity, AFDP uses NACKs to request retransmission of lost packets. Senders retransmit lost packets to the entire group. AFDP's rate-based flow control slows down senders every time they receive a NACK. Senders gradually increase their transmitting rate after every successful transmission.

AFDP allows sites to join and leave a distribution group by contacting the group's secretary, but does not provide a recovery mode to handle site failures.

2.2.3 Multicast File Transfer Protocol (MFTP)

In MFTP [8] the source may operate in unicast, multicast, or broadcast mode depending on what kind of information is being sent and what transmission modes are suppoted by the network. The MFTP client or receivers are able to handle any transmission mode.

The MFTP consists of group management and data transmission protocols. Using the data transmission protocol, the MFTP sender announces a file transfer session periodically during the announcement phase. In response to announcements, clients register to receive the file. MFTP sends data in passes. In the first pass, it sends the

entire file at a predefined transmission rate. In subsequent passes, the server transmits lost packets. At the end of each pass, the source requests status from receivers, who then respond in the form of ACKs if they have received all packets, or NACKs if packets have been lost. Receivers send ACKs, only if explicitly required by the source; otherwise they only send NACKs if they need packets to be retransmitted. The source may specify an address other than its own where receivers should send ACKs and NACKs. This is a way of aggregating receiver feedback to avoid implosion at the source. Feedback suppression is optional and is used with NACKs.

MFTP's group management protocol supports a public group to which announcements are sent. Data is transmitted on a private group address. While any client may join and listen to announcements on public groups, only clients authorized by the source may join the private group. MFTP uses two group management models open and closed. In closed groups, the source knows which clients are authorized to receive the transmission at announcement time. In open group model, the source does not know a priori who is allowed to receive the data. Any clients may request to participate in data delivery.

2.2.4 Reliable Multicast Transport Protocol (RMTP)

The RMTP [2] supports reliable one-to-many bulk data delivery. RMTP addresses feedback implosion by organizing group members into a control tree, which governs feedback propagation and processing. Intermediate nodes in the control tree of designated receivers (DRs) are responsible for buffering data received from the source, processing ACKs from their children, and retransmitting lost packets.

Receivers periodically unicast ACKs to their DRs informing them what packets have been correctly received. Depending on how many receivers lost the same packet, the DR decides whether it should multicast or unicast the retransmission. RMTP uses window-based flow control and defines a maximum transmission rate set at group establishment. RMTP allows dynamic group membership. Receivers may join and leave a multicast

group anytime during a session. However, knowledge of group membership is not required. RMTP not only allows receivers to join at anytime but also guarantees that they receive all the data reliably. This feature comes at the price of having the DRs buffer the data already sent throughout the session. RMTP's congestion control mechanism is based on shutting down the transmission window to one packet if the number of ACKs with retransmission requests exceeds a predefined threshold.

2.2.5 Tree-Based Multicast Transport Protocol (TMTP)

The goal of TMTP [3] is to provide reliable multicast communication support for one-to-many bulk data dissemination applications. For error and flow control, TMTP organizes group members into a control tree. Each subtree, or domain, is represented by its root, or the Domain Manager. Domains typically correspond to subnets. At group creation time, the maximum tree degree K (the maximum number of children at each level) is defined. The control tree grows and shrinks as members join and leave the group. Joining the control tree is based on an expanding ring search to locate the nearest domain manager whose degree is less than K.

A TMTP sender multicasts packets to the corresponding multicast group. The root of each subtree (including the data source) only waits for ACKs from its children to reclaim buffer space and advance its transmission window. An intermediate node does not have to wait for ACKs from all its children to unicast the ACK to its parent. ACK timeout triggers retransmission of the corresponding packet using limited-scope multicast. In addition to this sender-initiated error control approach, TMTP also relies on NACKs. TMTP receivers use limited-scope NACK multicast subject to supression.

TMTP uses a window-based flow control scheme and a predefined maximum transmission rate set at group creation time. TMTP supports dynamic group membership, where sites can join and leave a multicast group anytime during the life of a session. A session directory service provides primitives for creating, deleting, joining and leaving a multicast group. However, knowledge of group membership is not assumed.

2.3 Technical characteristics required in our multicast protocol

Remote Tutor requires a specific form of multicast delivery called dissemination. Dissemination involves $1 \times N$ communication in which a single sender must reliably multicast a significant amount of data to multiple receivers. Hence, the target application required in *Remote Tutor* is data dissemination. The *Remote Tutor* application is designed for WAN environment. The other requirement is that the flow control should be efficient like window based flow control. The final and most important need is that the protocol should be highly scalable in the WAN environment.

2.4 The selection of TMTP protocol

Reliable multicast transport protocols can be grouped into two application classes: multipoint interactive applications and data dissemination applications. Interactive and data dissemination services differ in their delivery delay(or delay variance) and reliability requirements. While interactive applications are willing to sacrifice reliability in favour of real-time delivery, data dissemination tools tolerate higher delivery delays but require reliable delivery.

The MDP [6] protocol could not be selected, since it does not have structure-based feedback control which limits its scalability, the most important requirement of our protocol. The AFDP protocol [7] proves to be good for LAN environment. But, it is not suitable for WAN applications and does not scale well. The MFTP [8] is a optimized protocol for file delivery rather attempting to be generalized reliable multicast transport protocol that can handle both file delivery and streaming data. This is a highly scalable protocol but is designed for LANs and does not suite for WANs. Hence, MFTP is also ruled out.

After an extensive study of the multicast transport protocols [1] grouped under

data dissemination [4] applications, the two protocols which seem to be suitable to our requirement were Reliable Multicast Transport Protocol (RMTP) [2] and Tree-Based Multicast Transport Protocol (TMTP).

The TMTP protocol [3] has been finally implemented. The reason for choosing TMTP as compared to RMTP is explained next. Both protocols require tree structure construction. In TMTP, the control is solely built on transport layer and does not require any support from IP multicast infrastructure. In contrast, RMTP requires IP multicast support for tree setup, which has to be done at the router. The selection of nodes for distribution of state management is done dynamically in TMTP. But RMTP makes static selection of such nodes depending upon the approximate location of the receviers. Hence, TMTP creates a balanced tree as the number of children in it is fixed for a node. But in RMTP, due to fixed nodes the tree may become unbalanced, overloading a node. The error control in TMTP uses ACKs and restricted NACKs as compared to RMTP which uses only ACKs. Therefore, TMTP achieves scalable and reliable dissemination [10] by reducing the processing bottleneck of sender-initiated approaches and avoids the long recovery times of the receiver-initiated approach.

Chapter 3

The TMTP protocol and modifications done in it

3.1 The TMTP protocol

The TMTP protocol [3] has the following features:

1. The TMTP uses IP multicast for routing of packets and their efficient delivery.
2. In TMTP a *hierarchical control tree* of dissemination group members is dynamically organized using an *expanding ring search*, as members join and leave a group.
3. In TMTP, flow and error control is distributed across several nodes. Thus, acheiveing scalability.
4. Error recovery is done by the receivers who use a combination of *restricted negative acknowledgement with nack suppression* and periodic positive acknowledgements. Also, care is taken to restrict the retransmission of packets to the required region.

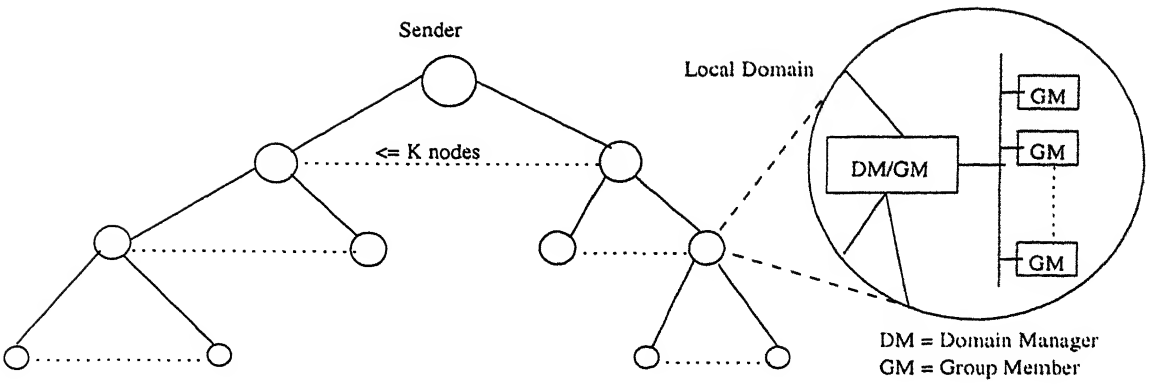


Figure 3.1: An example control tree with the maximum degree of each node $\leq K$.

In this protocol a sender process multicasts data on a dissemination group address. The sender process creates a dissemination group before it starts transmission on it. The protocol has no provision to ensure that the receiver processes are present before sender process starts.

The flow and error control management is done by creating a control tree on top of the transport layer. The tree construction is done by using expanding ring search i.e. TMTP organizes the group participants into domains or subnets. Each domain has a representative called Domain Manager (DM), who is responsible for the error recovery in that domain. The DMs can also provide error handling for the other DMs in the vicinity. For this purpose the DMs are organized into a control tree as shown in Figure 3.1

The sender serves as the root of the tree and can have a maximum of K children (DMs). The value of K is chosen at the time of tree creation and registration and does not include local group members in a domain. The local group members in a domain are called Group Members (GMs). The limited number of children (K) reduces the processing load on the sender as well as the internal nodes of the control tree. The protocol overhead thus grows as $Log_K(No\_of\_Receivers)$.

In TMTP, transmission of data is done by sender using the IP multicast. The transmission rate is controlled by using sliding window protocol. The sender and DMs are responsible for their respective domain's error recovery. The DMs send periodic

ACKs to their parents. Any receiver (DM or GM) can multicast NACKs in combination with NACK supression, as soon as it detects a packet loss during reception of packets.

The following sections describe the details of the TMTP protocol.

3.1.1 Group Management

The group management primitives are implemented in the other layers of *Remote Tutor* application. This is taken care of in signalling plane of *Remote Tutor*. The session directory is maintained at LMS. The LMS keeps the information about the ongoing sessions, the multicast address and port numbers of the ongoing sessions, the encryption key etc. The application joins the group after authenticating with the LMS. Authentication is followed by the other group management primitives like `CreateGroup`, `JoinGroup`, `LeaveGroup` and `DeleteGroup`.

The session directory provides the following group management primitives.

`CreateGroup(GName,CommType)`:

A sender creates a new group (with identifier `GName`) using the `CreateGroup` routine. `CommType` specifies the type of communication pattern desired. For example, here it is dissemination. If successful, `CreateGroup` returns an IP multicast address and a port number to use when transmitting the data.

`JoinGroup(Gname)`:

Processes that want to receive the data feed represented by `GName` call `JoinGroup` to become a member of the group. `JoinGroup` returns the transport level address (IP multicast address and port number) for the group which the new process uses to listen to the data feed.

`LeaveGroup(Gname)`:

Removes the caller from the dissemination group `GName`.

`DeleteGroup(GName)`:

When the transmission is complete, the sending process issues a `DeleteGroup` request to remove the group `GName` from the system. `DeleteGroup` also informs all participants, and domain managers that the group is no longer active.

3.1.2 Control Tree Management

In TMTP protocol, a control tree grows and shrinks dynamically by using the two operations, join tree and leave tree. Whenever a new DM is created it executes the Join tree protocol to become the member of the control tree. In the same way, if a DM which does not have any children can execute the leave tree protocol and leave the control tree.

3.1.2.1 Join tree protocol

The Figure 3.2 shows the join tree protocol which is executed by new DM created. A new domain manager begins an expanding ring search by multicasting a SEARCH_FOR_PARENT request message with a small time-to-live value (TTL). The small TTL value restricts the scope of the search to nearby control nodes by limiting the propagation of the multicast message. If the manager does not receive a response within some fixed timeout period, the manager resends the SEARCH_FOR_PARENT message using a larger TTL value. This process repeats until the manager receives a WILLING_TO_BE_PARENT message from one or more domain managers in the control tree.

All existing domain managers that receive the SEARCH_FOR_PARENT message will respond with a WILLING_TO_BE_PARENT message unless they already support the maximum number of children. The algorithm is shown in Figure 3.3.

The new domain manager then selects the closest domain manager (based on the TTL values) and directly contacts the selected manager to become its child. For each

```
let TTL = 1
While(Not Done){
   Multicast a SEARCH_FOR_PARENT msg
   Collect Responses
   If(no responses)
     Increment TTL  /* try again */
   Else
     Select closest respondent as parent
     Send JOIN_REQUEST to parent
     Wait for JOIN_CONFIRM reply
     If (JOIN_CONFIRM received)
          Not Done = False
     Else     /* try again */
 }
```

Figure 3.2: New Domain Manager Algorithm

```
Receive request message
If (request is SEARCH_FOR_PARENT)
   If (MAX_CHILDREN not exceeded)
      Send WILLING_TO_BE_PARENT msg
   Else
      / * do not respond * /
Else If (request is JOIN_REQUEST)
      Add child to the tree
      Send JOIN_CONFIRM msg
```

Figure 3.3: Existing Domain Manager Algorithm

```
If (I_am_a_leaf_manager)
   Send LEAVE_TREE request to parent
   Receive LEAVE_CONFIRM
   Terminate
Else  / * I am an internal manager */
   Fulfill all pending obligations
   Send FIND_NEW_PARENT message to children
   Receive FIND_NEW_PARENT reply from all children
   Send LEAVE_TREE request to parent
   Receive LEAVE_CONFIRM
   Terminate
```

Figure 3.4: Leave Tree Protocol, used after the last GM terminates

domain, its manager maintains a multicast radius for the domain, which is the TTL distance to the farthest child within the domain. The domain manager keeps the children informed of the current multicast radius. As described later in the description of the error control part of TMTP, both parent and its children in a domain use the current multicast radius to restrict the scope of their multicast transmissions. This reduces the traffic on the Internet. The multicast radius is updated whenever the farthest child is changed.

3.1.2.2 Leave Tree Protocol

A DM/Sender may have two types of children i.e. DMs or GMs. If a DM has only GM children then it is called *leaf manager*. In the other case, if a DM has DM children also along with GM children, then it is called *internal manager*. A internal manager cannot leave the tree unless all children DMs have left. The leave protocol is only for leaf managers. The Figure 3.4 shows the algorithm used to leave the control tree after the last local group member terminates.

This implementation provides "probably uninterrupted service" which means children of the departing manager continue to receive the feed while they reintegrate themselves into the tree. However, errors that arise during the brief reintegration time might not be correctable.

After a departing manager has fulfilled all obligations to its children and parent, the departing manager instructs its children to find a new parent. The children then begin the process of joining the tree all over again.

3.1.3 Delivery Management

TMTP is able to combine the advantages of sender-initiated and receiver-initiated approaches while avoiding their disadvantages. Logically, TMTP's delivery management protocol can be partitioned into three components: data transmission, error handling, and flow control. The following sections address each of these aspects.

3.1.3.1 The Transmission Protocol

The sender sends the data on the IP multicast address and it is received simultaneously by all DMs and GMs, in the control tree. The sender expects to receive positive acknowledgments in order to reclaim buffer space and implement flow control. However, to avoid the ack implosion problem of the sender-initiated approach, the sender does not receive acknowledgments directly from all the group members and, instead, receives ACKs only from its K immediate children.

Once a domain manager receives a multicast packet from the sender, it can send an acknowledgment for the packet to its parent because the branch of the tree the manager represents has successfully received the packet (even though the individual members may not have received the packet). That is, a domain manager sends ACK to its parent without waiting for ACKs from its own children. In addition, each domain manager periodically sends such ACKs, only to its parent.

3.1.3.2 Error Control and retransmission strategy

A TMTP traffic source (sender) requires periodic (unicast) positive acknowledgements and uses timeouts and (limited scope multicast) retransmissions to ensure reliable de-

livery to all its immediate children (domain managers). However, in addition to the sender, the domain managers in the control tree are also responsible for error control after they receive packets from the sender. Although the sender initially multicasts packets to the entire group, it is the domain manager's responsibility to ensure reliable delivery. Each domain manager also relies on periodic positive ACKs (from its immediate children), timeouts, and retransmissions to ensure reliable delivery to its children. When a retransmission timeout occurs, the sender (or domain manager) assumes the packet was lost and retransmits it using IP multicast within it's multicast radius.

TMTP also uses *restricted NACKs with NACK supression* to respond immediately to packet losses. When a receiver detects a packet loss, it waits for a random time and then sends a NACK. This NACK is restricted to a multicast radius and is send only if no other sibbling has send a NACK for this same packet. When a DM or Sender receives a NACK for a packet, it immediately retransmitts the missing packet.

3.1.3.3 Flow Control

TMTP achieves flow control by using a combination of rate-based and window-based techniques. The maximum rate is set when the group is created and never changes. TMTP's primary means of flow control consists of a window-based approach used for both dissemination from the sender and retransmission from domain managers. Within a window, senders transmit at a fixed rate.

TMTP's window-based flow control differs slightly from conventional point-to-point window-based flow control. Note that retransmissions are very expensive because they are multicast. In addition, transient traffic conditions or congestion in one part of the network can put backpressure on the sender causing it to slow the data flow. To oversimplify, TMTP avoids both of these problems by partitioning the window and delaying retransmissions as long as possible. This increases the chance of a positive acknowledgement being received and it also allows domain managers to rectify transient behavior before it begins to cause backpressure.

TMTP uses two different timers to control the window size and the rate at which the window advances. $T_{retrans}$ defines a timeout period that begins when the first packet in a window is sent. Since the transfer rate is fixed, $T_{retrans}$ also defines the window size. A second timer, T_{ack}, defines the periodic interval at which each receiver is expected to unicast a positive ACK to its parent.

The sender specifies the value of T_{ack} based on the Round trip time (RTT) to its farthest child. $T_{retrans}$ is chosen such that

$$T_{retrans} = n \times T_{ack}, \tag{3.1}$$

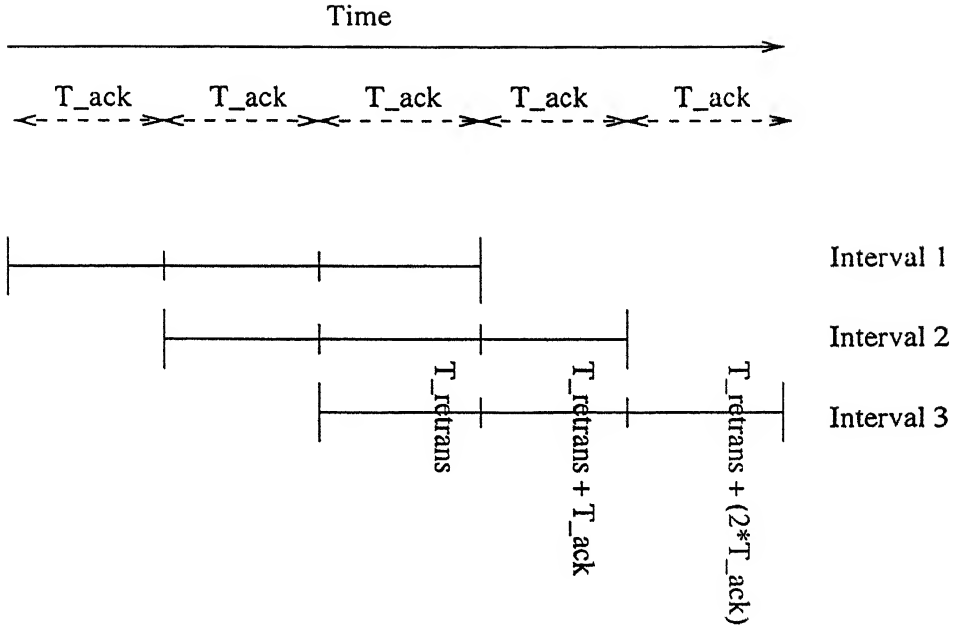
where, n is an integer, $n \geq 2$.

Both $T_{retrans}$ and T_{ack} are fixed at the beginning of transmission and do not change. A sender must allocate enough buffer space to hold packets that are transmitted over the $T_{retrans}$ period.

Figure 4.5 illustrates the windowing algorithm graphically. The sender starts a timer and begins transmitting data (at a fixed rate). Consider the packets transmitted during the first T_{ack} interval. Although the sender should see a positive ACK at time T_{ack}, the sender does not require one until time $T_{retrans}$. Instead, the sender continues to send packets during the second and third interval.

After $T_{retrans}$ amount of time, the timer expires. At this point, the sender retransmits all unACK'd packets that were sent during the first T_{ack} interval. Retransmissions continue until all packets in the T_{ack} interval are acknowledged at which point the window is advanced by T_{ack}. On the receiving end, packets continue to arrive without being acknowledged until T_{ack} amount of time has expired.

A domain manager must continue to hold packets in its buffer until all of its children have acknowledged them. If the children fail to acknowledge packets, the domain manager's window will not advance and its buffers will eventually fill up. As a result, the domain manager will drop and not acknowledge any new data from the sender, thereby causing backpressure to propagate up the tree which ultimately slows the flow of data.



Three retransmission intervals where T_retrans = 3 * T_ack

At the end of first interval, packets sent during the first T_ack period are retransmitted. At the end of the second interval, packets sent during the second T_ack period are retransmitted. At the end of the third interval, packets sent during the third T_ack period are retransmitted.

Figure 3.5: Different stages in sending data

There are three reasons for using multiple T_{ack} intervals during a retransmission timeout interval ($T_{retrans}$). First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. Second, a larger retransmission interval gives receivers sufficient time to recover missing packets using receiver-initiated recovery when only one (or a few) packets in a window are lost. This avoids unnecessary multicast retransmissions of a window full of data. Third, multiple T_{ack} intervals during the retransmission interval provide sufficient opportunity for a domain manager to recover from transient network load in its part of the subtree without unnecessarily applying backpressure to the sender.

We have chosen the value of the multiplying factor n to be 3 based on empirical evidence [3]; the appropriate value depends on several factors including expected error rates, variance in RTT, and expected length of the intervals with transient, localized congestion. Further study is necessary to determine whether or not value of n should be chosen dynamically using an adaptive algorithm. Secondly, what should be the algorithm ?

3.1.4 Performance Measures and other criteria in TMTP

The two measures that evaluates the performance of the protocol are : *end-to-end delay* and *processing load.* To measure the end-to-end delay, we required that each receiving application send back a single positive acknowledgment (a GOT_IT message) to the sending application when the entire data transmission was complete. The sending application then calculated the end-to-end delay as the time between the beginning of the transmission and the time at which the last group member's final GOT_IT message is received.

To measure the processing load at the sender, receivers, and domain managers, we need to monitor processing activities like receiving and processing a selective positive acknowledgment. Again, since it is difficult to measure the amount of processing time

needed for such activity (and highly dependent on the operating system and architecture), we can choose to simply count the total number of such events at the sender to estimate the processing load generated by a protocol.

Yavatkar Rajendra *et. al* [3] mention about the various experiments performed to test the TMTP protocol. The test is carried out using a general sender-initiated reliable multicast transport protocol and TMTP protocol with same set of receivers and environment. It shows that TMTP protocol is better in performance.

3.1.4.1 Impact of data size

As the file size increases, the number of packets transmitted increases, thereby increasing the number of events (such as ACK/NACK processing or timer events) that affect the processing load at the sender (or a domain manager). Similarly, end-to-end delay is likely to increase due to more time needed to deliver all the packets.

In the case of TMTP, the processing load has a small increase because the work is distributed among many nodes in the control tree. Consequently, the sender does not have to process acknowledgments or retransmission requests from all the receivers. TMTP's end-to-end delay rises at a significantly lower rate than that of a general protocol. This occurs because error recovery in TMTP proceeds concurrently in different parts of the control tree rather than sequentially.

3.1.4.2 Impact of the Group Size

For a general Reliable multicast transport protocol there is a sharp increase in processing load with increase in number of receivers because the sender solely shoulders the responsibility for processing acknowledgments and retransmission requests (or timeouts) from each receiver. In the case of TMTP, the processing load at the sender (and each domain manger) is limited by the maximum number of immediate children in the control tree and, therefore, shows almost no increase as the number of receivers is

```
Receive NEW_DM packet and queue it
Compare the timestamp of the packets received
If more than one packet with least timestamp
    Wait for delay timer to expire
    Send a NEW_DM packet
Else
    Packet with least timestamp becomes NEW_DM
If New DM's IP Address is equal to my IP Address
    Call DM alogorithm  / * I am new DM */
Else
    Change IP Address of Parent equal to New DM's
            / * New Parent's IP Address * /
```

Figure 3.6: New DM selection algorithm

increased.

3.2 Modification of TMTP protocol

Most of the protocol is taken as it is, except some slight changes as mentioned below.

3.2.1 Parent Alive Information

The sender and DM are required to send a heartbeat packet to their children at regular intervals to show that they exist. If this packet stops reaching, the children know that they have lost their parent. Then, depending upon whether it is a DM or GM, they can execute their respective recovery protocols. If it is a DM, it will start all over again by sending SEARCH_FOR_PARENT packet. In case of GM, a New DM selection algorithm is executed, which is described next.

3.2.2 New DM selection Algorithm

The TMTP's leave tree protocol is for the case when there is no GM child. Hence, the parent cannot leave unless GM children have left. In the modified protocol, the case

with GM children is considered seperately. The New DM selection algorithm is shown in Figure 3.6.

The protocol is executed by all the GMs in a domain as they detect the loss of their parent. The GMs send a New DM packet with a random delay and suppression. If a GM is about to send a new DM packet, receives such packet from another GM, then it supresses its own packet and queues the received packet. After a fixed interval, the packet with least timestamp is considered and its sender becomes the new DM in that domain. If more than one new DM packet has the same least timestamp, then the process is again repeated.

Chapter 4

Design details of the protocol

The protocol is designed to work as a layer between application and transport layers. The design of the program can be understood from the conceptual diagram of TMTP shown in the Figure 4.1. The protocol is designed such that it has a single control which is transfered to various modules on a timely basis. The basic concept here is to use a single thread by running a single loop in which we check whether there is a packet received or there is some action to be taken. Hence, there are only two functions to be done. When neither the packet is there nor any action to be taken, the process goes to sleep waiting for first action in the action queue to be scheduled.

The action to be taken is implemented by creating an action queue. The members in this queue have a time field and an action type. In the loop, the action queue is checked for the next action to be taken. When the next action gets scheduled, depending upon the type of action, the control is switched to a particular module as shown in the Figure 4.1. The modules may be to send data or heartbeat packet. The other actions may include initializing a receiver or searching a new DM. When a particular action has to be taken after some time interval, an action is inserted into the action queue with the corresponding action type and the time at which the action has to be taken. The other function called inside the loop, is to check for a received packet. As soon as a packet is received, it is passed for processing to a yet another function.

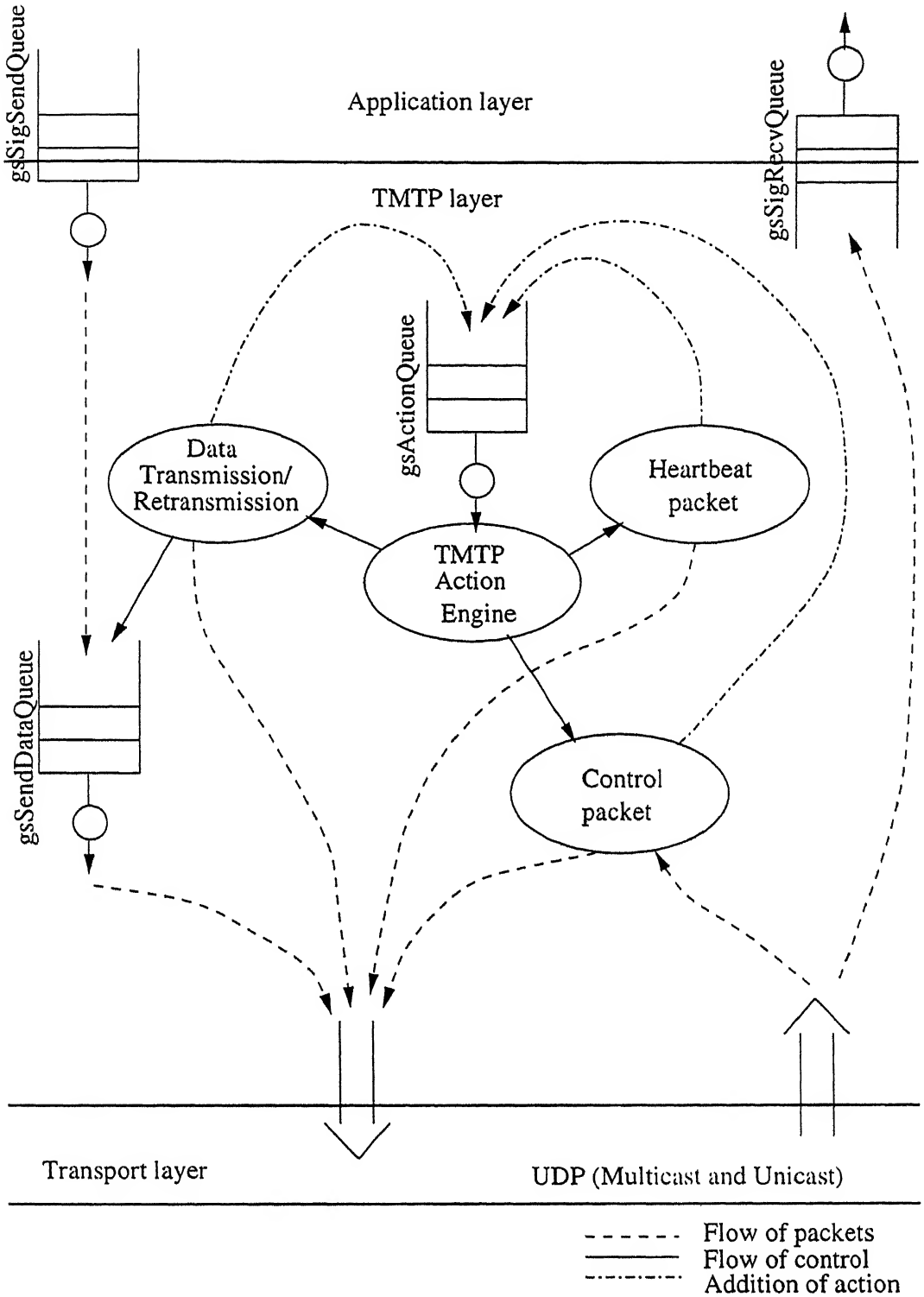


Figure 4.1: Conceptual diagram of TMTP protocol implementation

4.1 Various data structures

4.1.1 Record

Each module sender, DM or GM has its own requirement of data stuctures for keeping certain information. Sender needs to keep record of its children, their status, TTL distance and index. Similarly, a child has to keep data related to its parent.

Parent Record: In the parent record the details such as the parent's IP address, port number and its multicast radius is available. The structure *gsParentInfo*, of the parent record is given in the Appendix A.

Child Record: It is an array of child record structures. Each child record contains the details of a child. The Appendix A gives the structure *gsChildRecord* for a child describing its details. The global variable *giStudentPresent* gives the total number of children present at any time.

4.1.2 Structure

There is a structure *StNetPackInfo* used for handling send and received data from the UDP socket. It gives the data, its size and address information in a, easy to access single structure. The structure *LstStNetPackInfo* forms a node of link list of above type of structures. The details of these structures are given in Appendix A.

4.1.3 Queues

During the process of data transmission packets are to be buffered till an acknowledgement is received. This can be done by maintaining data queues for the data. There are five queues for different uses in the program. The details of these queues can be had from the Appendix A.

Action Queue: This queue *gsActionQueue*, grows as the actions are inserted into it based on the time at which it has to be worked upon. The deletion from this queue is done as soon as the time field of a node becomes less than the current time.

Send Queue: This is the interfacing queue *gsSigSendQueue*, between the application layer and the TMTP layer as shown in the Figure 4.1. The data to be send using the TMTP protocol are queued into this queue.

Receive Queue: This queue *gsSigRecvQueue*, is also on the same interface, but this time it is in the reverse direction. The data packets received from the network are processed i.e. the TMTP header removed and the extracted data is queued in this queue.

New DM Queue: This queue is formed when a GM's parent is lost. The GMs multicast a New DM packet. Such New DM packets received in a given interval are compared for the least timestamp to select a New DM for the domain.

Data buffer Queue: The queue *gsSendDataQueue* is used to buffer the data packets, whose acknowledgements have not yet arrived. The retransmissions of data packets are done from this queue.

4.2 Packet format description

For making various packet formats for TMTP, help is taken from the RMTP packet formats [9]. The TMTP protocol uses fifteen different types of packet for the data transmission and for sending control information to its peer process. Two different packets may have same formats but have different information in certain field. The IP multicast address and port number are included in many of the packet formats. This is done keeping in view the future modifications in the protocol. The protocol may use more than one multicast address, one for control packets and another for data packets. We have used one address only. The Figure 4.2 lists the various packets with their corresponding TYPEs and transmission used.

S.No.	Name of Packet	Packet type	Transmission Multicast(M)/Unicast(U)
1	Data transmission	1	Multicast
2	Data retransmission	2	Multicast
3	Acknowledgement	3	Unicast
4	Negative Acknowlegement	4	Multicast
5	Join Request	5	Unicast
6	Join Confirm	6	Unicast
7	Search_for_parent	7	Multicast
8	Leave Request	8	Unicast
9	Leave Confirm	9	Unicast
10	New DM selection	10	Multicast
11	Heartbeat	11	Multicast
12	Congestion control	12	Unicast
13	Willing_to_be_parent	13	Unicast
14	RTT Measure	14	Unicast
15	RTT Acknowledgement	15	Unicast

Figure 4.2: Table listing the various types of packets used in TMTP

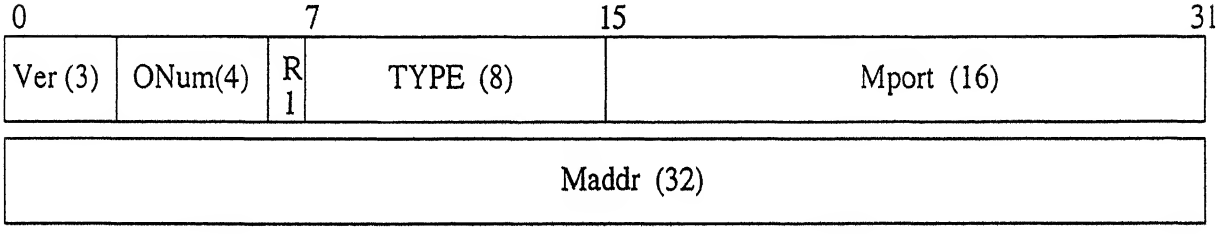


Figure 4.3: TMTP header packet format

4.2.1 The TMTP header

The TMTP header is a common part of all the packets. Each packet has the TMTP header at the beginning of the packet. The TYPE field of the header describes the type of packet. The Figure 4.3 shows the format of the TMTP header.

VER: Specifies the version of the protocol.

ONum: It is a number which specifies the number of option present in the packet. Presently, this field is not used.

TYPE: Specifies the type of packet.

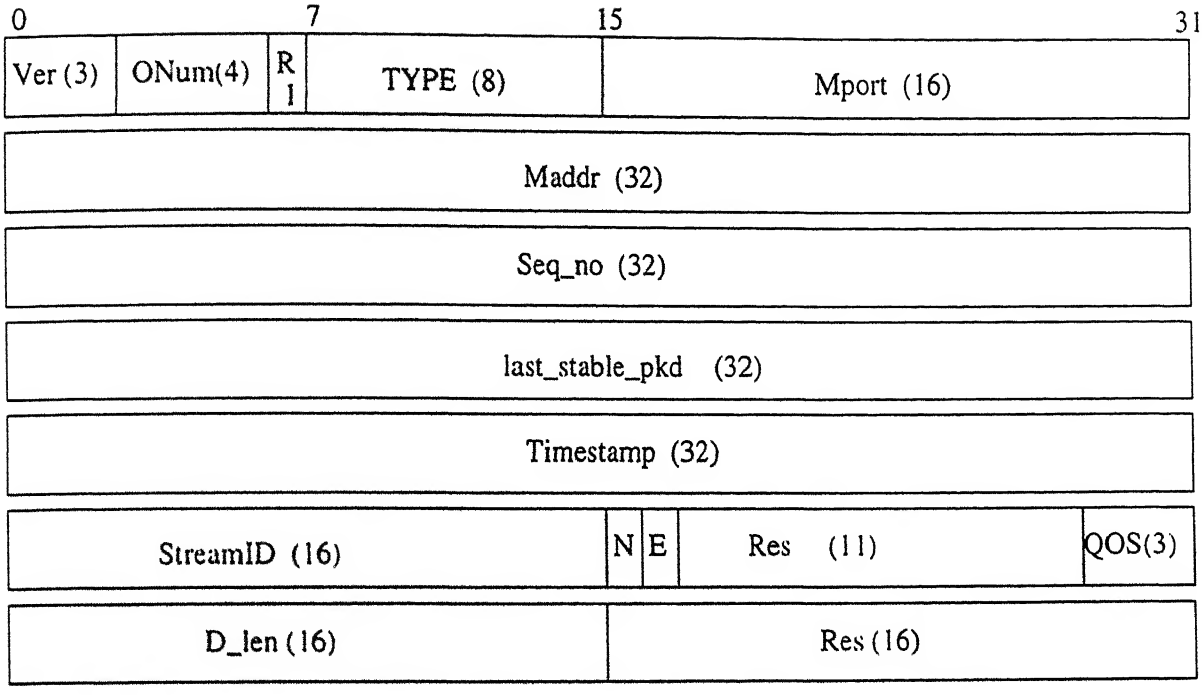


Figure 4.4: Data packet format

Mport: Gives the port on which transmission takes place.

Maddr: Gives the multicast address on which transmission takes place.

R or Res: This field are reserved bits for future use. This field may be present in other packet formats also.

4.2.2 The Data/Retransmission packet

This Data packet is multicast by the sender and the retransmission packet is retransmitted by DM in their local region. The packet format is shown in the Figure 4.4.

seq_no: A sender stamps the data packet that it sends with a sequence number. The sequence number are 32 bits and have a valid range from 2 to $2^{32} - 1$. The sequence number 0 and 1 are not used.

last_stable_pkd: The field contains the sequence number of the highest numbered stable packet at the sender. If the lowest numbered data packet in the data queue

becomes stable, i.e. all the children have received the packet, then the Sender deletes the packet from the data queue. The Sender receives stability information from the stable sequence number of the ACKs it receives. The sender initializes last_stable_pkd as StartSequenceNumber-1. last_stable_pkd is the sequence number of the last stable packet which the sender has removed from its data queue. All lower numbered packets are stable. The sender cannot retransmitt any packet having sequence number less than equal to last_stable_pkd.

Timestamp: This is a 32 bit timestamp which gives the time when the packet was formed.

StreamID: This is a 16 bit ID which uniquely identifies the stream. This is not implemented at present.

N: NACK enable flag, not used presently.

E: This is a flag that is set to 1 to indicate the last packet of the stream.

QOS: This field may be used to specify the desired quality of service for the data packet.

D_len: This is the length of the data in bytes.

4.2.3 The ACK/NACK packet

The ACK packet is a unicast packet, periodically send by every child (DM) to its parent. The NACK packet is multicast by a DM or GM. The Figure 4.5 shows the format of this packet.

Timestamp: This is a 32 bit timestamp which gives the time when the packet was formed.

M_Group_Addr: This is the multicast group address for sending NACKs. In TMTP only one address is used.

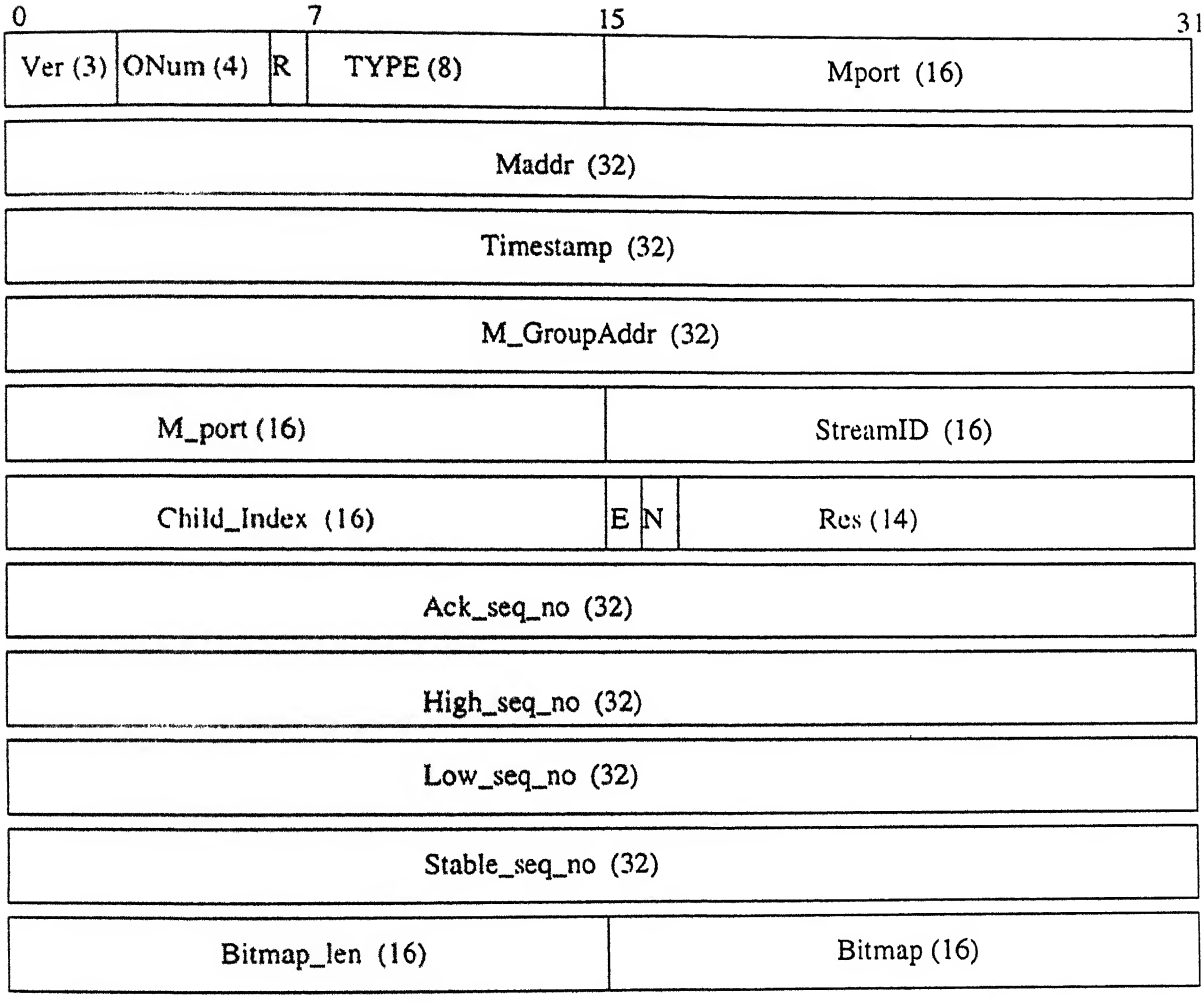


Figure 4.5: ACK/NACK packet format

Mport: This the port number correponding to above address.

StreamID: This is a 16 bit ID which uniquely identifies the stream. This is not implemented at present.

Child_Index: This is the index number of the child node, transmitting ACK packet, in its parent's child list.

E: This flag is set to 1 for the last ACK, to indicate all packets have been received.

N: This flag is set to 1 to indicate that this is a NACK packet. In ACK packet it is set to 0.

Ack_seq_no: This is the sequence number for which NACK is being send. For ACK packet it is set to 0.

High_seq_no: This is the sequence number of the highest numbered packet received.

Low_seq_no: This is the sequence number of the lowest numbered missing packet.

Stable_seq_no: The stable sequence number is the sequence number of the highest contignous stable packet received.

Bitmap_len: This is the length of the bitmap in 16 bit words.

Bitmap: The bitmap of missing packets between low sequence number and high sequence number.

4.2.4 The Join Request packet

The request to join a sender/DM parent is made by a DM child through this packet. The Figure 4.6 shows the packet format.

Org_TTL: This is the multicast radius of the child requesting the join.

SeqNo: This specifies the number of times this request has been made.

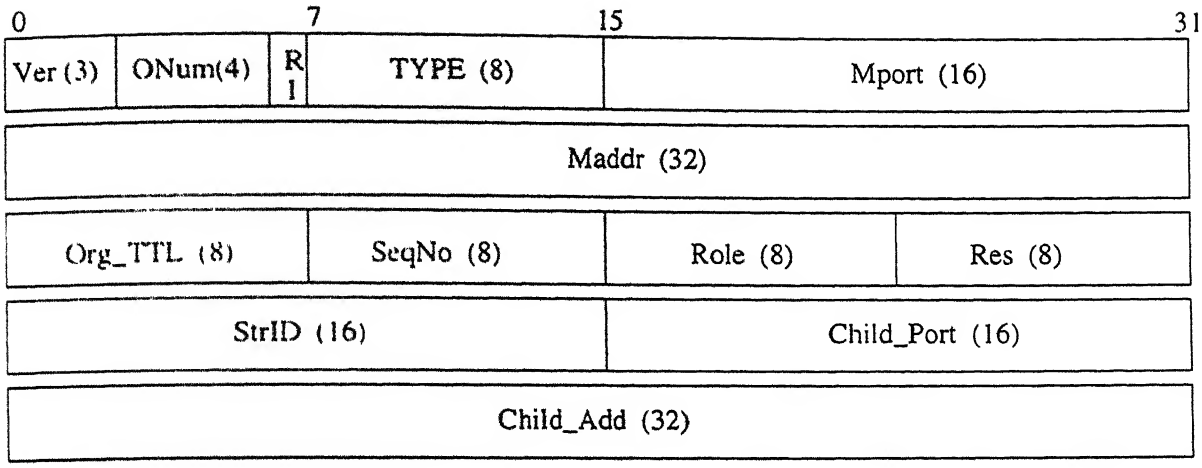


Figure 4.6: Join request packet format

Role: It is the role of the requesting child, which is DM here.

StrID: This is a 16 bit ID which uniquely identifies the stream. This is not implemented at present.

Child_Port: Specifies it's own unicast port number.

Child_Add: Specifies it's own IP address.

4.2.5 The Join Confirm packet

It is the packet send to a child in response to its join request packet. The Figure 4.7 shows the packet format.

Org_TTL: This field is not in use currently.

Index: This is the child's index in the parent's child record.

Parent_Add: This is the parent's IP address.

Parent_Port: This is the parent's port number.

Net_TTL: This is the parent's multicast radius.

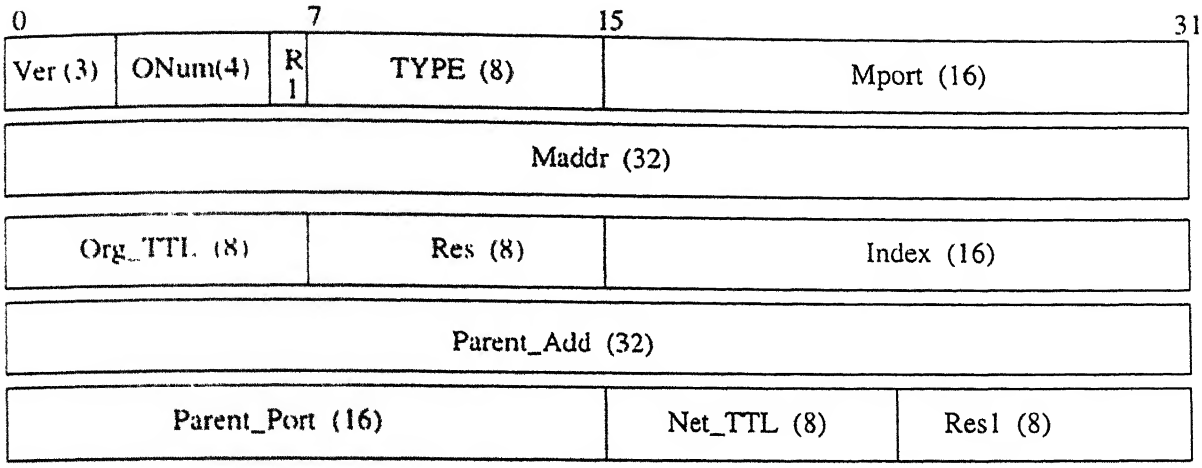


Figure 4.7: Join Confirm packet format

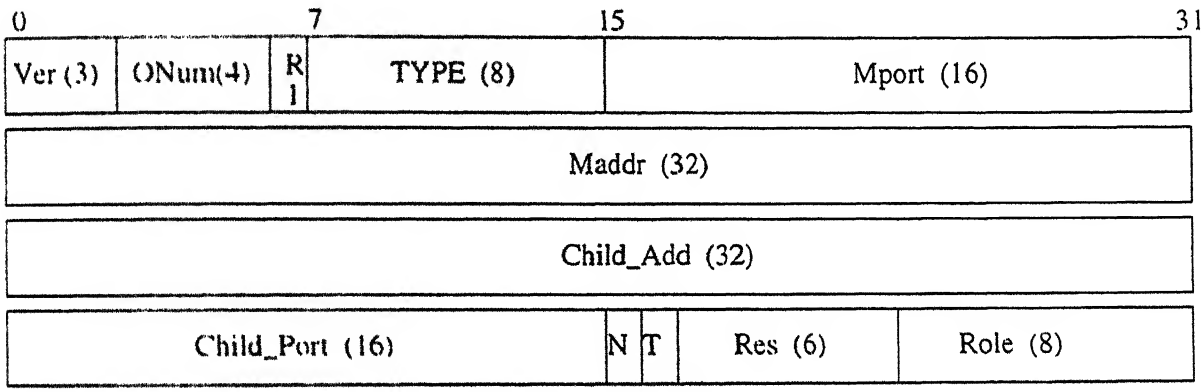


Figure 4.8: Search Parent packet format

4.2.6 The Search Parent packet

This packet is send as soon as a process starts initialization as a receiver. This packet is multicast periodically, with increments in TTL, untill a response is received. The packet format is shown in Figure 4.8.

Child_Add: This is the receiver's own IP address.

Child_Port: This is the receiver's port number.

N,T: These bits are for future use.

Role: This field specified whether the child is a DM or GM.

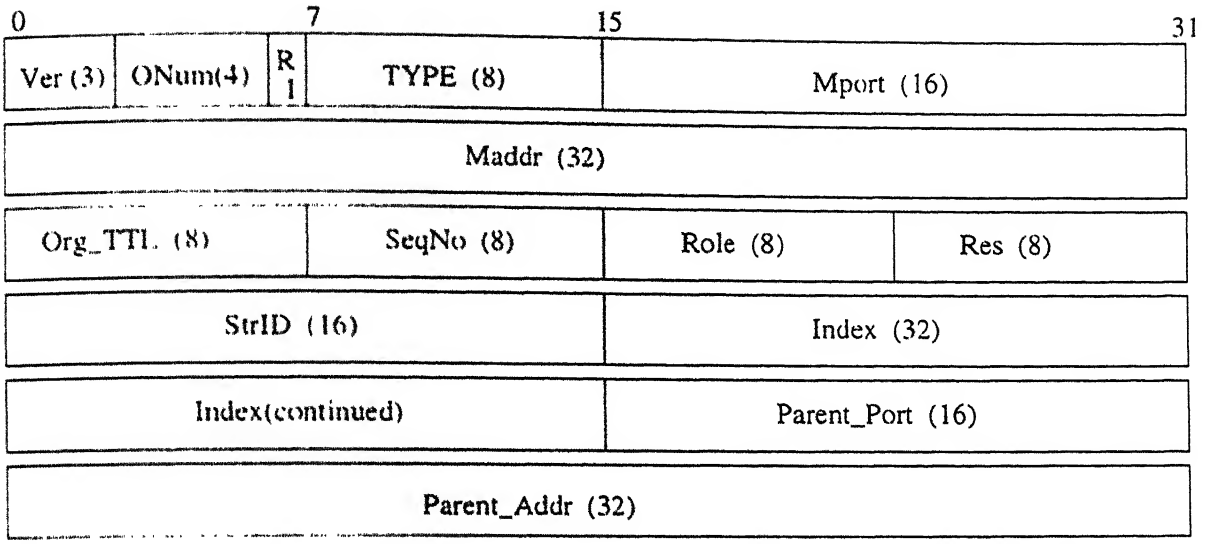


Figure 4.9: Leave Request packet format

4.2.7 The Leave Request packet

This packet is send by child (DM) to its parent to request for leave. The Leave request packet format is shown in Figure 4.9.

Org_TTL: This is the TTL radius of the requesting child.

SeqNo: This field gives the number of times, the request has been made.

Role: It is the role of the requesting child, which is DM here.

StrID: This is a 16 bit ID which uniquely identifies the stream. This is not implemented at present.

4.2.8 The Leave Confirm packet

This packet is send by the parent to the child to confirm the child's termination from the control tree. The Figure 4.10 shows the packet format for this type of packet.

Index: The parent sends the child index in this field.

StrID: This is a 16 bit ID which uniquely identifies the stream. This is not imple-

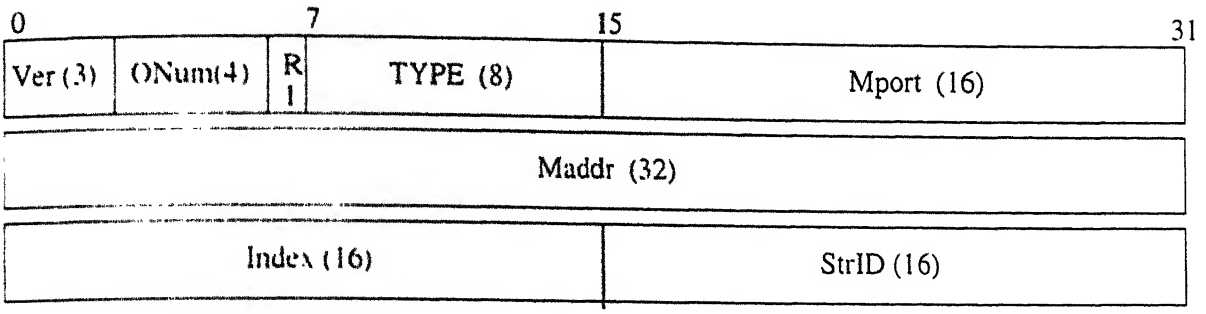


Figure 4.10: Leave Confirm packet format

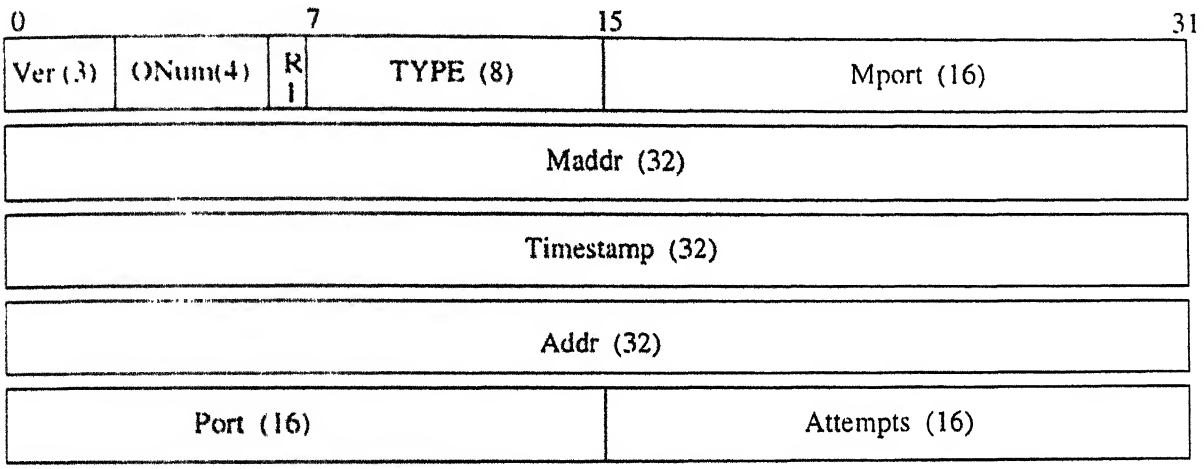


Figure 4.11: New DM packet format

mented at present.

4.2.9 The New DM packet

This packet is multicast by the GMs to select a new DM among the GMs in their domain. The Figure 4.11 shows the packet format adopted for this packet.

Timestamp: This is a 32 bit timestamp which gives the time when the packet was formed.

Addr: This is the IP address of the GM, who is sending the packet.

Port: This is the port number of the GM, who is sending the packet.

Attempts: This is the number of times the New DM packets have been send.

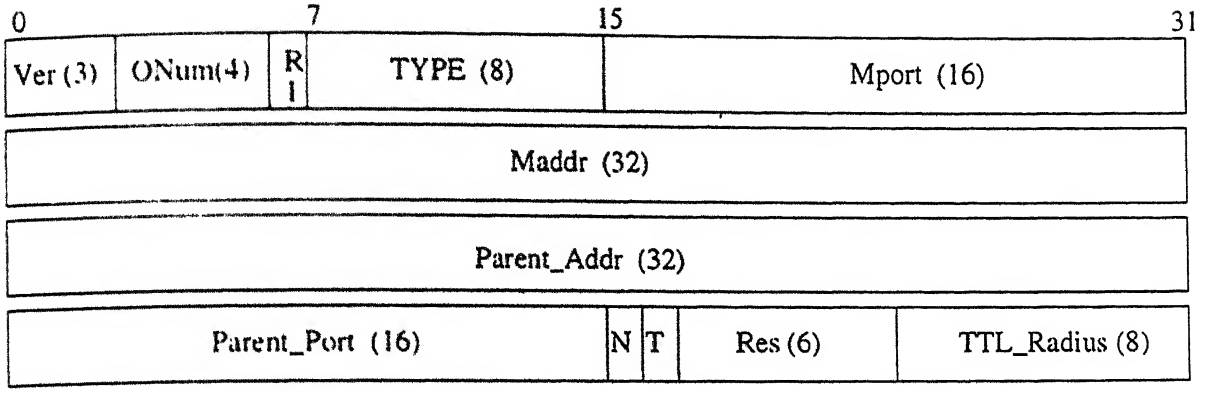


Figure 4.12: Hearbeat packet format

4.2.10 The Heartbeat packet

This packet is used to show the existence of a parent. It is send periodically by the parent (Sender or DM). The packet format of heartbeat packet is shown in the Figure 4.12.

Parent_Addr: It is the IP address of the parent.

Parent_Port: It is the port number used by the parent for transmission.

N,T: These bits are for the future use.

TTL_Radius: It is the current multicast radius of the parent. Any changes are automatically updated in databases of children.

4.2.11 The Congestion control packet

This packet is used to inform the Sender about the congestion in the network. The format of this packet is shown in the Figure 4.13.

Loss Rate: This field gives the packet loss rate at a receiver.

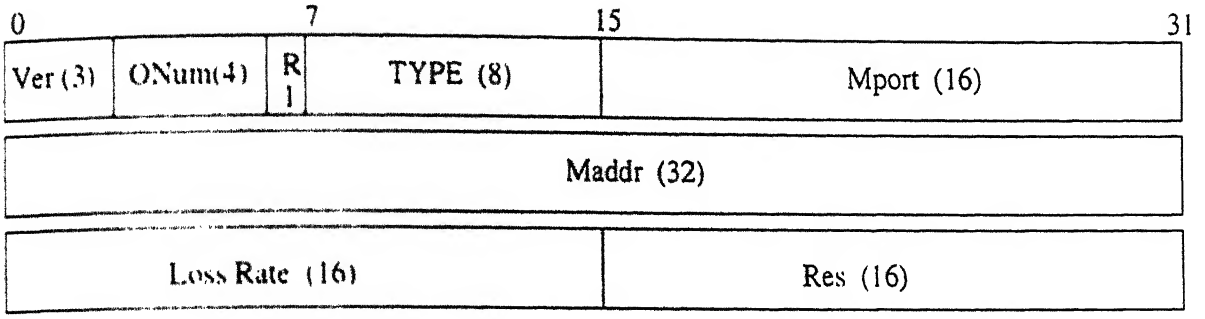


Figure 4.13: Congestion control packet format

4.2.12 The Willing Parent packet

This packet is send by a parent in response to a search packet. The Figure 4.14 shows the packet format for this type.

Org_TTL: Specifies the multicast radius to the parent.

R: This flag is set to 1, to indicate a rejoin request.

Role: Sends the child's role, extracted from the search packet.

No of Children: This gives the total children of that parent.

Sen1_Maddr: Specifies the parent's IP address.

Sen1_Mport: Specifies the parent port number used for transmission.

Sen1_StrID: This is a 16 bit ID which uniquely identifies the stream. This is not implemented at present.

The other fields in this packet are for future use.

4.2.13 The RTT Measure/Acknowledgement packet

This packet is used to measure RTT (Round Trip Time) for a given receiver. The Figure 4.15 shows the details of this packet.

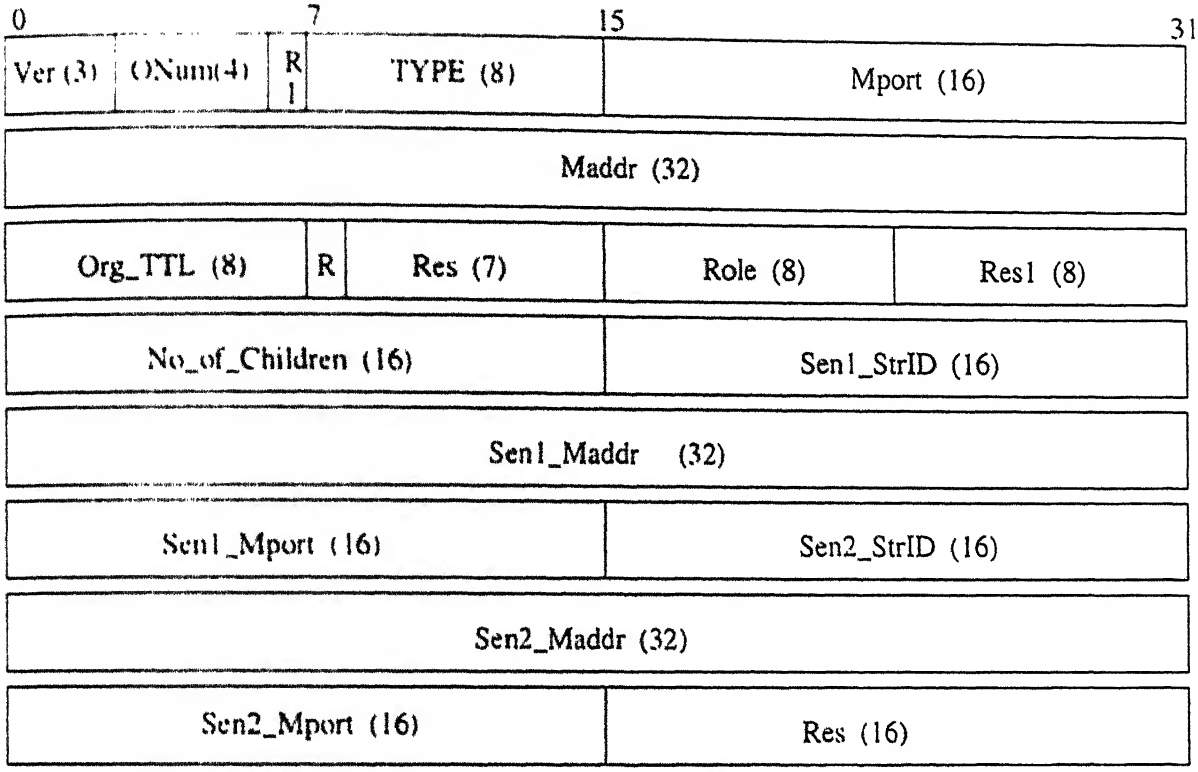


Figure 4.14: Willing Parent packet format

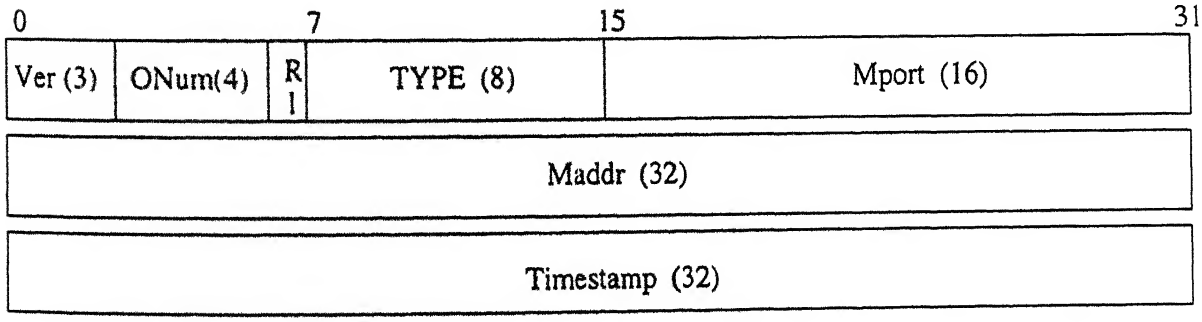


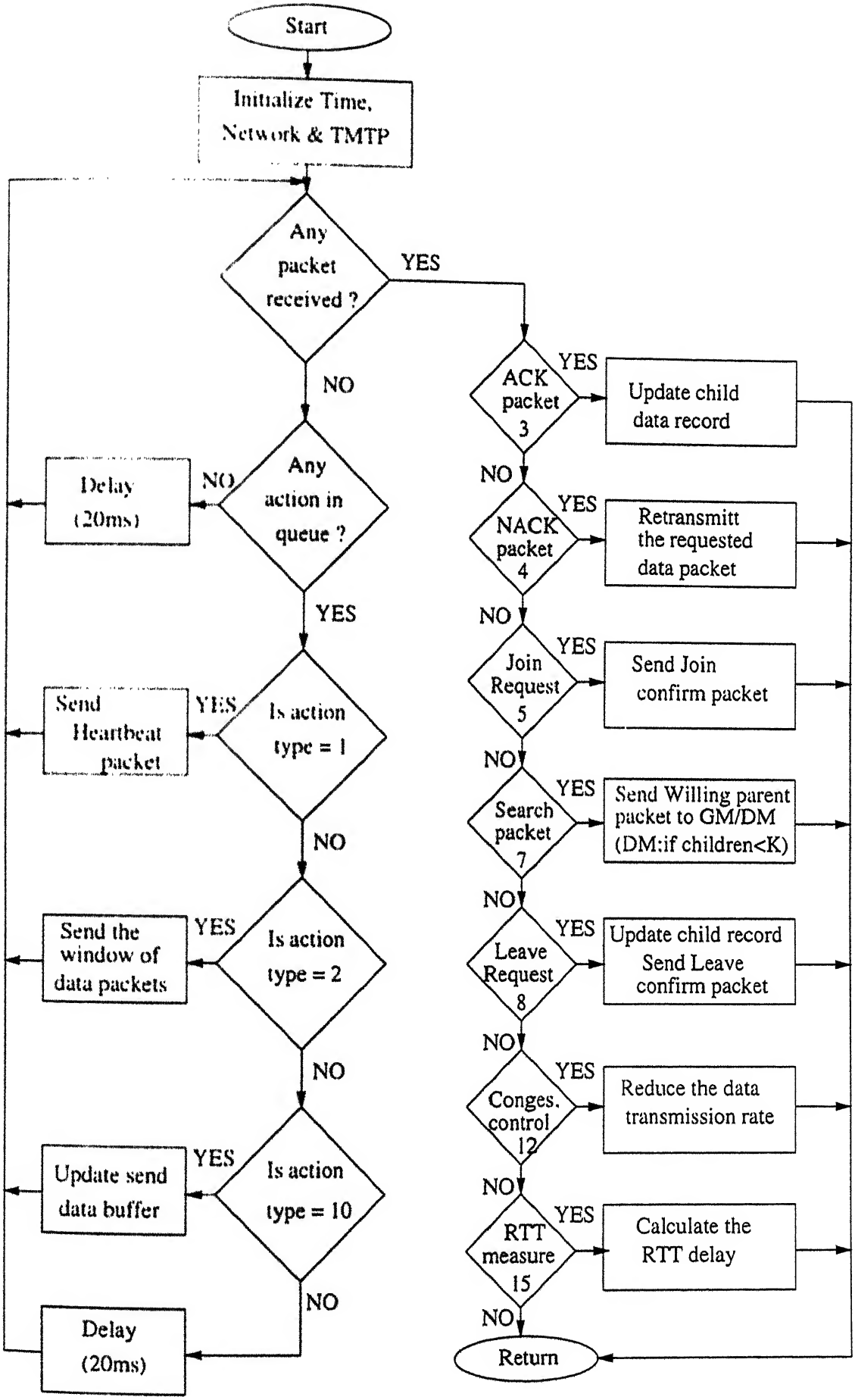
Figure 4.15: RTT Measure/Acknowledgement packet format

Timestamp: This field gives the time of the instant at which this packet is send.

Only the TYPE field in the header differenciates the two packets.

4.3 Flow diagrams of Sender, DM and GM

There are four diagrams in this section, three for the modules Sender, DM and GM. There is one more flow diagram showing the general flow of control for a receiver initialization.


Start
Initialize Time, Network & TMTP
Any packet received ?
YES
NO
Any action in queue ?
NO
Delay (20ms)
YES
Is action type = 1
YES
Send Heartbeat packet
NO
Is action type = 2
YES
Send the window of data packets
NO
Is action type = 10
YES
Update send data buffer
NO
Delay (20ms)
ACK packet 3
YES
Update child data record
NO
NACK packet 4
YES
Retransmitt the requested data packet
NO
Join Request 5
YES
Send Join confirm packet
NO
Search packet 7
YES
Send Willing parent packet to GM/DM (DM:if children<K)
NO
Leave Request 8
YES
Update child record Send Leave confirm packet
NO
Conges. control 12
YES
Reduce the data transmission rate
NO
RTT measure 15
YES
Calculate the RTT delay
NO
Return

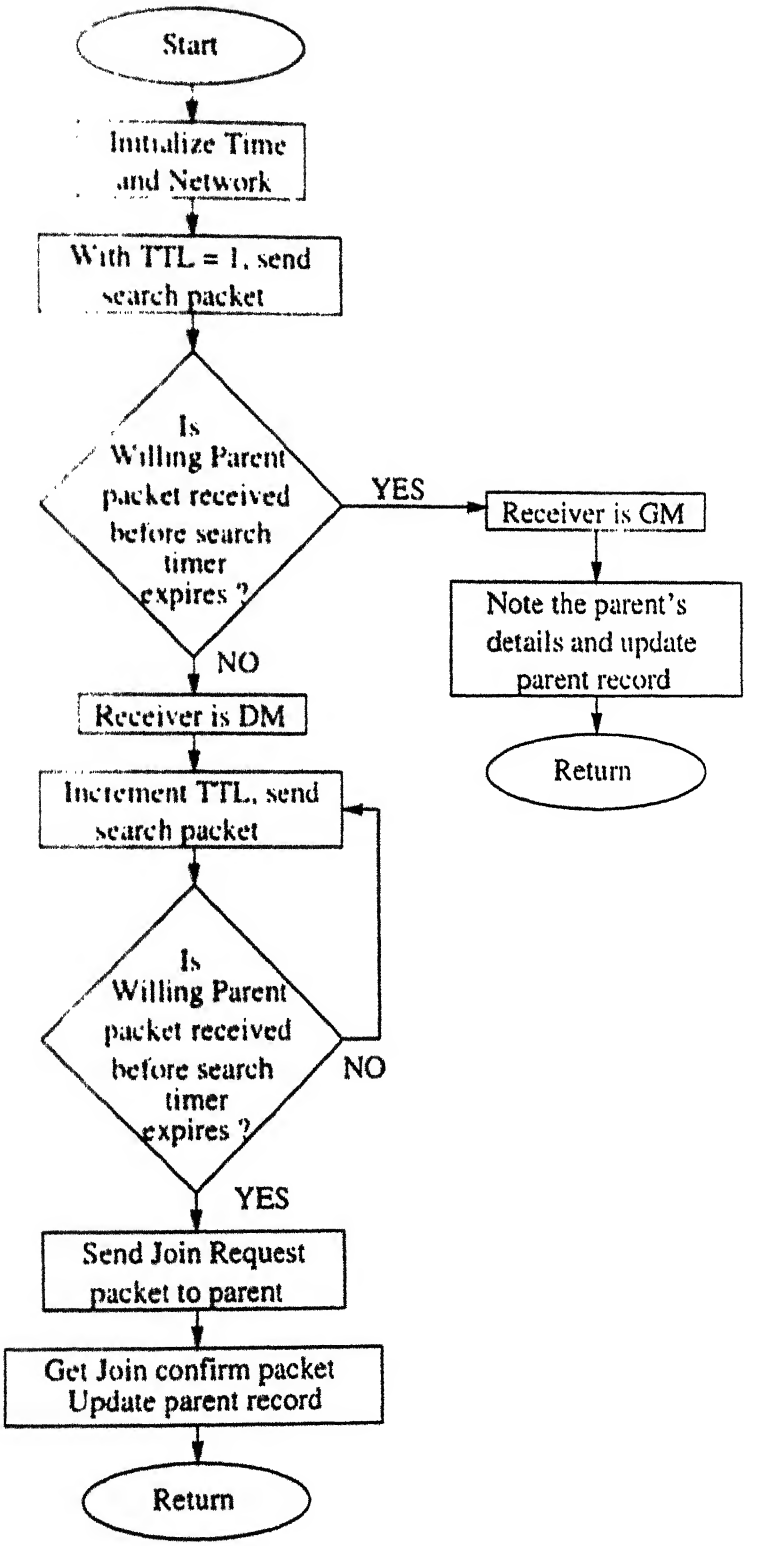
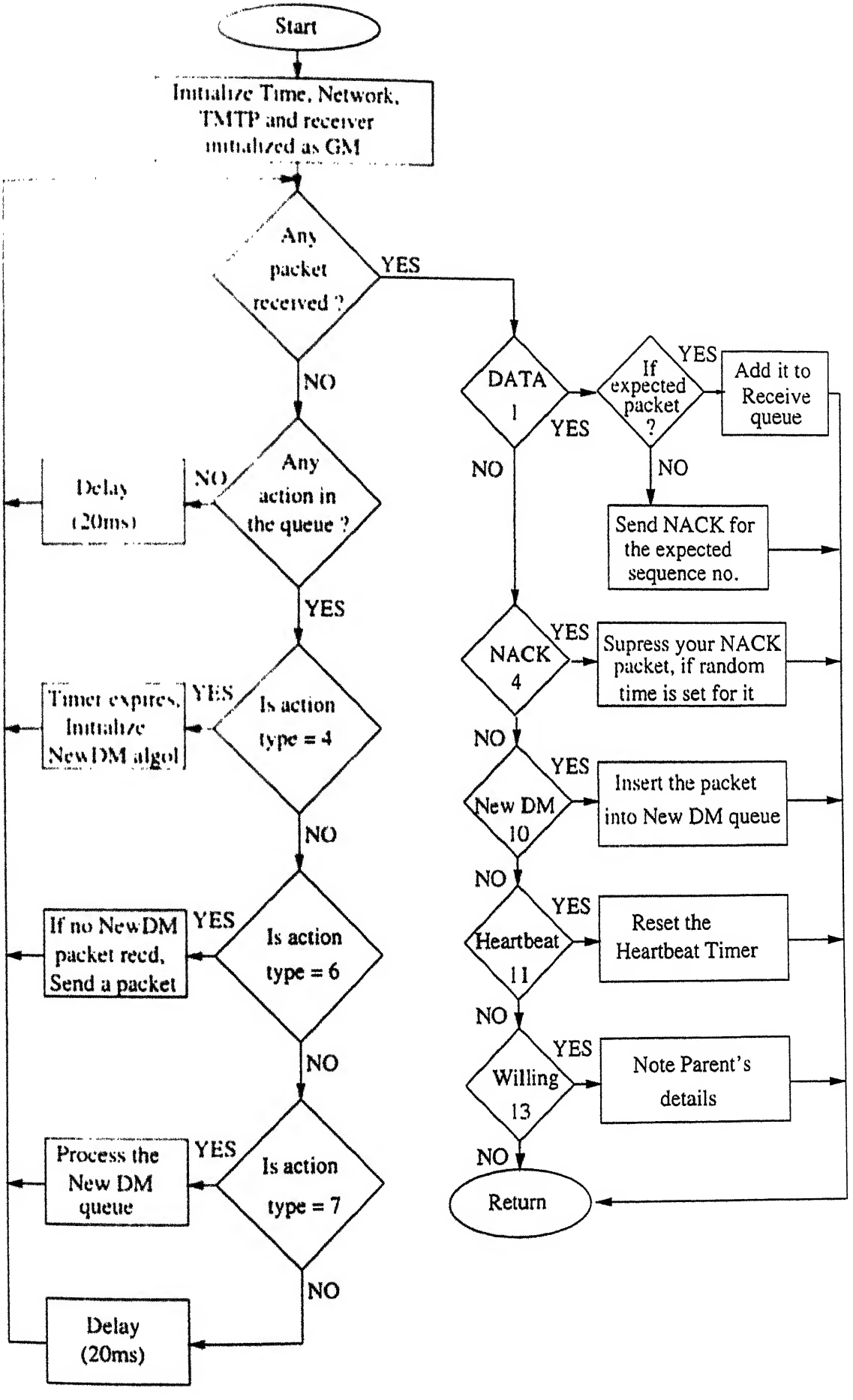
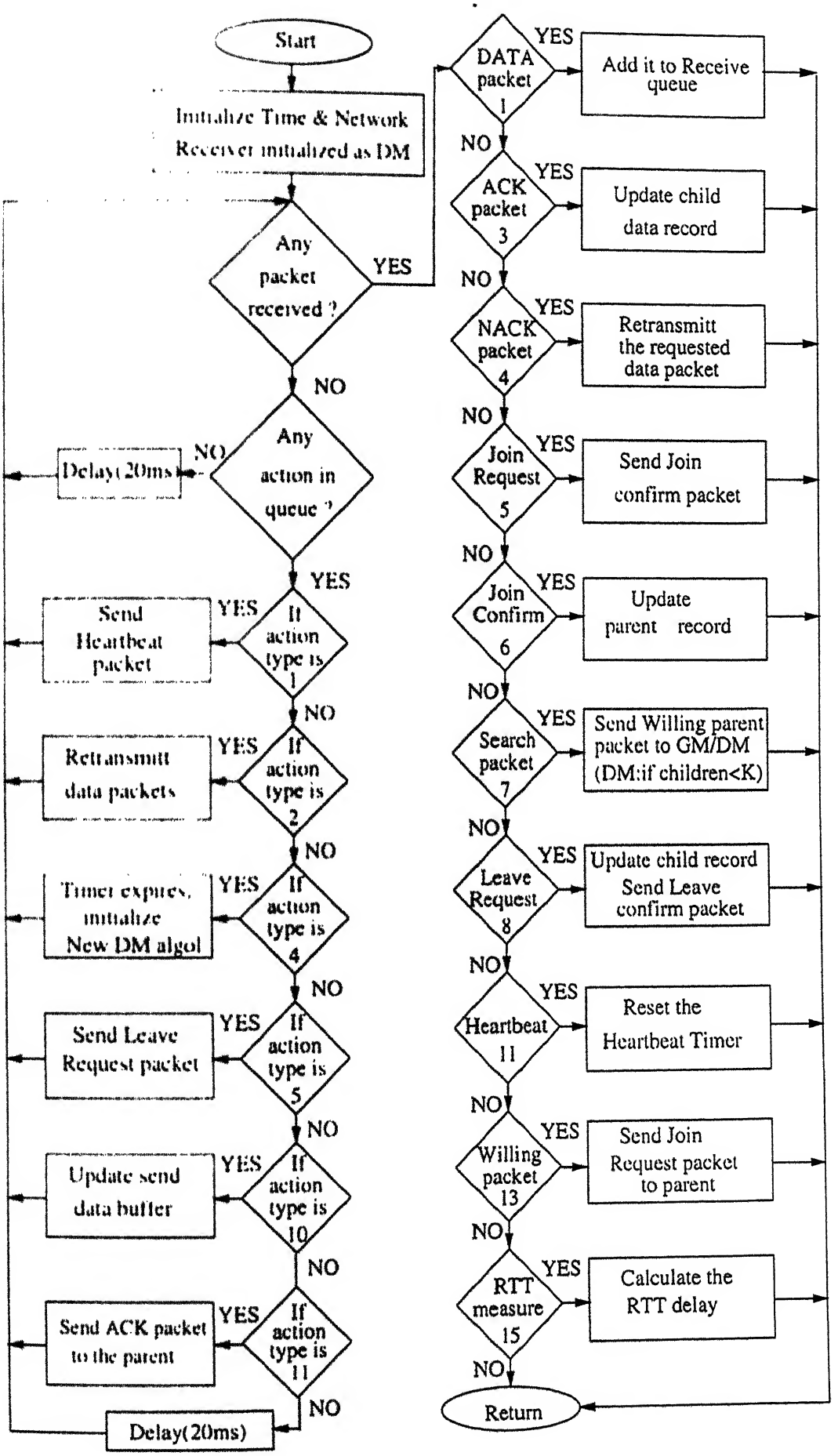


Figure 4.17: Flow diagram of Receiver Initialization


Start
Initialize Time, Network, TMTP and receiver initialized as GM
Any packet received ?
YES
NO
DATA 1
YES
If expected packet ?
YES
Add it to Receive queue
NO
Send NACK for the expected sequence no.
Any action in the queue ?
NO
Delay (20ms)
YES
NACK 4
YES
Supress your NACK packet, if random time is set for it
NO
Is action type = 4
YES
Timer expires, Initialize NewDM algol
NO
New DM 10
YES
Insert the packet into New DM queue
NO
Is action type = 6
YES
If no NewDM packet recd, Send a packet
NO
Heartbeat 11
YES
Reset the Heartbeat Timer
NO
Willing 13
YES
Note Parent's details
NO
Is action type = 7
YES
Process the New DM queue
NO
Delay (20ms)
Return


Start
Initialize Time & Network
Receiver initialized as DM
Any packet received ?
YES
NO
Any action in queue ?
NO
Delay(20ms)
YES
If action type is 1
YES
Send Heartbeat packet
NO
If action type is 2
YES
Retransmitt data packets
NO
If action type is 4
YES
Timer expires, initialize New DM algol
NO
If action type is 5
YES
Send Leave Request packet
NO
If action type is 10
YES
Update send data buffer
NO
If action type is 11
YES
Send ACK packet to the parent
NO
Delay(20ms)
DATA packet 1
YES
Add it to Receive queue
NO
ACK packet 3
YES
Update child data record
NO
NACK packet 4
YES
Retransmitt the requested data packet
NO
Join Request 5
YES
Send Join confirm packet
NO
Join Confirm 6
YES
Update parent record
NO
Search packet 7
YES
Send Willing parent packet to GM/DM (DM:if children<K)
NO
Leave Request 8
YES
Update child record Send Leave confirm packet
NO
Heartbeat 11
YES
Reset the Heartbeat Timer
NO
Willing packet 13
YES
Send Join Request packet to parent
NO
RTT measure 15
YES
Calculate the RTT delay
NO
Return

Chapter 5

Implementation details and Testing

5.1 Implementation details

A common source code is written for all the three modules i.e. Sender, GM and DM. When the application calls the TMTP routine, it can set the flags corresponding to required module (Sender, GM or DM). The main functions used in the application are described below under different sections, grouped according to the type of section.

5.1.1 Initialization Functions

These functions are called during the start of the process.

`InitTime()`

The first thing done by the starting process is to set a reference time for the whole session. The reference time is used for calculating the relative time at any instant during the program execution.

`InitNet()`

This function initializes the multicast socket on which TMTP packets will be send

and received. The multicast address and port number is initialized. The multicast options like loopback and TTL are also set as our requirements.

`InitTMTP()`

This function initializes TMTP, i.e. the various TMTP queues and data structures in it are assigned with initial values. Since a single source code is written for all the three modules, the initialization specific to a module is carried out in this function. The data rate and heartbeat packet rate are set and also the total number of children for a parent (K) is assigned. For receivers, the receiver initialization function is called next, inside this function. For Sender, the heartbeat and data delivery functions are initialized.

`InitReceiver()`

This routine sets the receiver process as a GM or DM, by sending search packets. Every time when this function is called, it sends a search parent packet with a increment in TTL value. It then adds an action in the action queue for sending the next search packet after TSEARCH interval.

`InitNewDMAlgol()`

This function initializes the new DM algol by adding an action in the action queue to send a new DM packet after a random time. It also adds an action for new DM packet processing, after TNEWDM interval.

5.1.2 Core Function

The `RunCore()` function is the heart of the whole source code. The function executes a loop in which it checks whether there is a received packet or an action to be taken. The details of this function can be understood by referring to the first paragraph of the chapter four.

5.1.3 Socket Functions

These set of routines are to initialize a socket for multicast sender or receiver . It also contains routines to send and receive packets over multicast and unicast links. It contains routines to set destination address before sending a packet.

`InitSigSocket()`

This function initializes the signalling socket and returns the socket handle for use by the other routines. If there is an error during socket creation or binding, it returns with the particular error message.

`SendSPack()`

This function takes a structure *StNetPackInfo* as a parameter. The data is send through the UDP socket to the address given in the structure.

`RecvSPack()`

The main purpose of this function is to listen to a socket for an incoming data. The function returns with NULL, if no data is received. If a data is received, a *StNetPackInfo* structure is filled up with the received information and its pointer is returned.

`SetSrcAddr()`

This function is used to extract the information about a data source. The parameters are passed to get the details in them.

`SetDestAddr()`

This routine fills the information about the destination address in the structure *StNetPackInfo*, before sending it through the socket. The parameters passed are the address and port number to which the data is to be send.

5.1.4 Interfacing Functions

The routines given in this section are those which are called by the application layer. Hence, they serve as the interfacing functions between the application and TMTP layer.

`SendSigPack()`

This function is called by the application layer whenever it wants to send a non real-time data. The message to be send is given in buffer and the message size should not exceed the maximum buffer size. The function returns -1, if the size exceeds the maximum buffer size. The function returns 0, if the *gsSigSendQueue* is full. If these two conditions does not occur, then a data packet is formed and queued and a 1 is returned to show the success.

`RecvSigPack()`

This function is used by the application layer. It returns with packets on the control port that can be handled by the main thread, running on the application layer. It requires pointer to information buffer and pointer to a integer variable, for the size of the buffer. It returns the queue size of receive buffer *gsSigRecvQueue*. It returns -1, if the queue is empty.

`Close_TMTP()`

This function is also used by the main thread on the application layer. The application calls this function, when it wants to stop the TMTP process. This may further initiate the leave protocol for a DM child.

5.1.5 Packet transmission and Processing

This section includes the routines which are used to transmitt data and to receive and process the received data.

`Sendfunc()`

This function checks for data packets in *gsSendDataQueue* and new packets in *gsSigSendQueue*. Then depending on the send window size, transmitts some packets on the multicast address. The above is true for a TMTP sender. For the case of DM parents, there will be only retransmission queue. It then, adds an action, in the action queue for sending next set of data packets after TSEND interval of time.

`recvdata()`

This function is called whenever a data packet is received and is passed as a parameter to it. The function queues the packet into the *gsSigRecvQueue* queue, if it is not full. If the queue is full, it returns 0. Thus, the received packet is lost due to full buffer. This function is called inside the `M_Recr()` function when a data type packet is received.

`M_Recr()`

This routine is called by the `RunCore()` function, when a packet is received and has to be processed. The function checks for the type of packet and accordingly, further processing is carried out.

5.1.6 Tree Management Functions

This section contains the funtions which make different types of packets and send to the required address.

`Heartbeat()`

This function creates and sends a heartbeat packet. It then adds an action in the action queue for sending the next heartbeat packet after a THBEAT interval.

`Send_ACK_Pkd()`

This function creates a ACK packet and sends it to the parent. This function is called after every TACK period.

`Send_NACK_Pkd()`

This function creates a NACK packet for the sequence number passed to it as a parameter. It then sends the packet within a multicast radius of its parent after a random delay.

`Send_Join_Req_Pkd()`

This function creates a join request packet and sends it to the parent node.

`Send_Join_Confirm_Pkd()`

This function creates a join confirm packet and sends it to the child, whose index is passed as an argument to it.

`Send_Leave_Req_Pkd()`

This function sends a leave request packet to the parent node.

`Send_Leave_Confirm_Pkd()`

This function sends a leave confirm packet to the child, whose index is passed as an argument to it.

`Send_Willing_Pkd()`

This routine sends a willing parent packet as a response to a received search packet. The argument of this function gives the details of the child who sent the search packet. If it is a DM, then the function sends the willing packet only if the number of children under it has not exceeded the maximum number.

`Send_NewDM_Pkd()`

This function creates a new DM packet and sends it on the multicast address within its domain i.e. with TTL equals to 1.

5.1.7 Utility Functions

The set of functions in this section are serving as utilities in our implementation.

`Curr_Time()`

This routine calculates the relative time from a reference time. The reference time is initialized during the start of the process. The function returns the relative time in milliseconds.

`Rand_Time()`

This function uses a random number which is within some bound and adds it to the current time. The sum is then returned by this function as a random time. Such random time is used where a packet has to be send after a random time delay.

`memalloc()`

This function takes the size of the memory as an argument and allocates the memory of that size. Then, it returns a pointer to that memory location.

`ErrMsg()`

This function is used to print the error message on the screen. The message is passed as a parameter to it.

The various time variables used in the implementation are listed in the Figure 5.1. The list also gives the functions of each of the variables.

5.2 Testing details

The software was tested on multiple subnets connected through routers supporting multicast routing. The Figure 5.2 gives the diagram of the test bed used for rigourous testing of the software. The source code is initialized for various modules (Sender, DM

S.No.	Time variable	Function of the variable
1.	TSEND	Inverse of data transmission rate .
2.	THBEAT	Inverse of heartbeat packet rate .
3.	TSEARCH	Time interval between search packets.
4.	TACK	Inverse of ACK packet transmission rate.
5.	TNEWDM	Time interval between New DM packets.

Figure 5.1: Table to define the various time variables

and GM) and run on different PCs on the network.

There are four subnets in the network shown which are connnected with three routers. A total of six PCs are present in the complete network. The basic assumption of the *Remote Tutor* application is that the Sender is the first module which is started. The other modules start later on. The various tests performed are mentioned below.

5.2.1 DM and GM setup

Once a Sender program is started on PC1, then dynamic tree construction starts. In the subnet1, when PC2 starts initializing, it becomes a GM; since it finds a parent in the domain itself. In case of subnet2, let us assume that PC3 is booted up first. Hence, as it could not find a DM in that domain, it itself becomes a DM of that domain. After becoming a DM, it then searches for a parent. Finally, finding PC1 (Sender) as its parent. Proceeding in a similar way, PC5 and PC6 setup as DMs of their respective domains. Once they setup as DMs, they will search for parents and depending on which of them booted up first, may lead to a consruction of a control tree as shown in Figure 5.3.

5.2.2 DM and GM leave

The next testing which was carried out was for a entity which wants to leave the tree. Since we have different algorithms for GMs and DMs, they were tested seperately. If

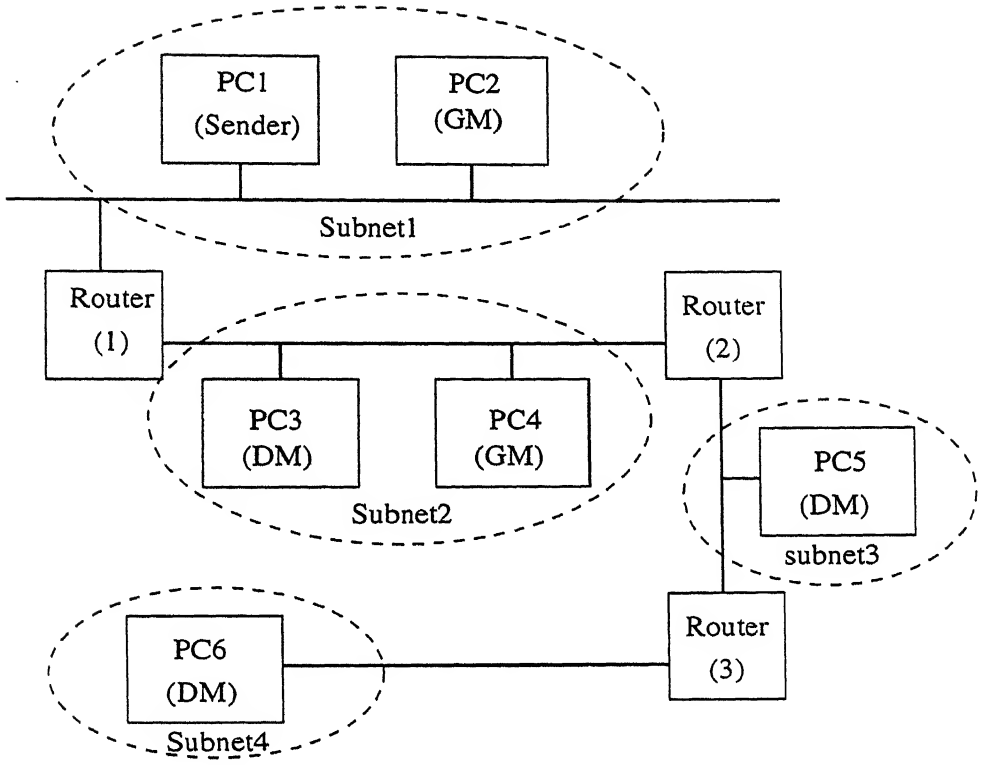


Figure 5.2: The test bed

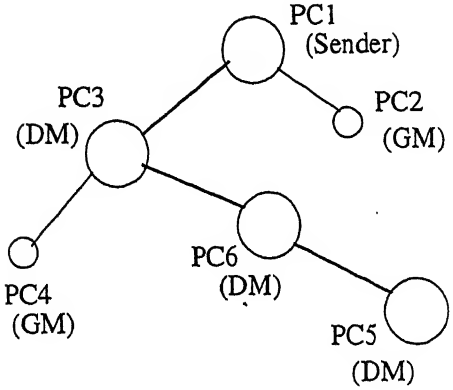


Figure 5.3: An example of tree construction

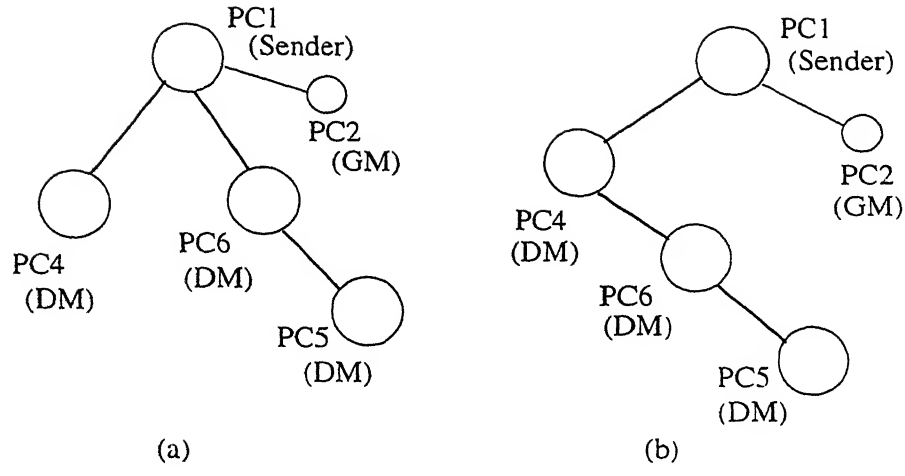


Figure 5.4: An example of tree rearrangement, when the nodes leave

a GM wants to leave, it does not have to carry out any communication with its parent before leaving. It just leaves without informing its parent.

In case of DM, for example, PC3 in Figure 5.3, the DM gets the leave command from its upper layer. As soon as it gets this message, it would stop sending heartbeat packets. When this DM's children i.e. PC4 and PC6 notices that their parent is lost, they start their individual recovery mechaninsms. The PC4 (GM) will execute the new DM algorithm within the domain. As there is no other GM in this domain, so PC4 will itself become a new DM and start the search for parent. In its search it may find PC1 (Sender) as its parent.

For the case of PC6 (DM), it will start search for its parent as soon as it discovers that it lost its parent. The PC6 (DM) may find PC1 (Sender) again. as a parent resulting in a changed tree, as given in Figure 5.4 (a). After PC3 (DM) has left the control tree, the tree configuration may become anything, it does not depend directly on the network. For example, in the case above, if PC4 becomes DM before PC6 can find a parent; then it may happen that PC4 becomes parent of PC6, resulting in yet another tree configuration as shown in Figure 5.4 (b).

5.2.3 Reliability of data transmission over lossy network

The reliability of the network is tested by intentionally dropping some of the packets at the receiver. The packets are dropped with probability p called as the *probability of packet loss*. For this, a special routine was inserted in the receive function, where lossy channel was simulated by dropping the packets. There is a random number generator, which generates a uniformly distributed random number between 0 and 1. Whenever a packet is received, a number is generated. If this number is less than p, where p is the packet loss probability and is equal to 0.01, then the packet is dropped, otherwise it is processed. This value of p equal to 0.01, indicates that one packet in every 100 packets is lost.

The retransmissions occured and end-to-end reliability was achieved. A more rigourous testing requires simulation of a real Internet with a large number of receiving nodes. This would have allowed us to observe the packet implosion problem. Thus, it can be said that the test we performed was mainly the test of functionality and not performance.

Chapter 6

Conclusion and future work

6.1 Conclusion

The software has been implemented on Window95 using VC++ version 5.0. The source code is written in C language. It has been developed in steps, i.e. with continuous testing process carried out in parallel throughout the development.

The program has been tested for all the three modules Sender, DM and GM, in the Ernet and Networks Group's laboratory, IITK, using a maximum of three routers in the network. The performance with such an environment has been found to be satisfactory.

6.2 Future work

The future testing of the software is required to be done in a more larger network like WANs. Since the measurement of peformance measures like end-to-end delay and processing load can only be carried out in such an environment. The software can then be further optimized depending on the results of the measurements.

Appendix A

Various Data structures

A.1 Structures

```
struct StNetPackInfo {
        char buffer[MAX_PACKET_SIZE];
        int size;
        struct sockaddr_in;
}

struct LstStNetPackInfo {
        struct StNetPackInfo *packinfo;
        struct LstStNetPackInfo *next;
}

struct StActionQueue {
        u_int32 time;
        int type;
}

struct StChildRecord {
```

```
        int Index;
        int flag;
        u_int8 Net_TTL;
        char Addr[ADDR_SIZE];
        u_int16 Port;
}

struct StSendData{
        u_int32 SeqNo;
        u_int32 Timestamp;
        int ReXmitt_Count;
        struct StNetPackInfo *packinfo;
        struct StSendData *next;
}

struct StParentInfo{
        int Index;
        int Net_TTL;
        char Parent_Add[ADDR_SIZE];
        u_int16 Parent_Port;
}
```

A.2 Queues

```
struct LstStNetPackInfo *gsSigSendQueue, *gsSigRecvQueue;
struct StSendData *gsSendDataQueue;
struct StActionQueue *gsActionQueue;
int giStudentPresent;
```

References

[1] Katia Obraczka, "Multicast Transport Protocols: A Survey and Taxonomy", *IEEE Communications Magazine*, January 1998.

[2] Sanjoy Paul, Krishan K. Sabnani, John C. H. Lin and Supratik Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)", *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 3, April 1997.

[3] R. Yavatkar, J.Griffioen and M. Sudan, "Reliable Dissemination Protocol for Interactive Collaborative Applications", *Proc. ACM Multimedia '95*, 1995.

[4] "Reliable Multicast Links", http://research.ivv.nasa.gov/RMP/links.html, T. Montgomery, Oct. 1997.

[5] Don Towsley, Jin Kurose and Sridhar Pingali, "A Comparision of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols", *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 3, April 1997.

[6] J. Macker and W. Dang, "The Multicast Dissemination Protocol (mdp) Framework", Internet draft, IETF, draft-macker-mdp-framework-00.txt, Nov. 1996.

[7] Jeremy R. Cooperstock, Steve Kotsopoulos, "Exploiting Group Communication for Reliable High Volume Data Distribution", http://www.ecf.utoronto.ca/afdp.

[8] K. Miller *et. al*, "Starburst Multicast File Transfer Protocol (mftp) Specification", Internet draft, IETF, draft-miller-mftp-spec-02.txt, Jan. 1997.

[9] B. Whetten *et. al*, "The RMTP-II Protocol", Internet draft, IETF, draft-whetten-rmtp-ii-00.txt, April 1998.

[10] Brian Neil Levine and JJ Garcia-Luna-Aceves, "A Comparision of Known Classes of Reliable Multicast Protocols", ftp://ftp/cse.ucsu.edu/pub/tr.

[11] Andrew S. Tanenbaum, *Computer Networks*, Third Edition, Prentice Hall of India Private Ltd.

[12] W. Richard Stevens, *UNIX Network Programming*, Prentice Hall of India Private Ltd.

[13] Robert L. Kruse, Bruce P. Leung, Clovis L. Tondo, *Data Structures and Program Design in C*, Prentice Hall of India Private Ltd.

[14] Byron S. Gottfried, *Schaum's outline of Theory and problems of programming with C*, Second Edition, McGraw-Hill Publishing Company Ltd.

[15] Manoj Gupta, *SlideTalk: Slide Display Implementation* M.Tech. Thesis, Deptt. of Electrical Engg. I.I.T. Kanpur, 1997.

[16] Sudheer Kumar N. V., *Remote Tutor: Organizing, Recording and Playback of Lectures* M.Tech. Thesis, Deptt. of Electrical Engg. I.I.T. Kanpur, 1998.

A 130810

Date Slip

This book is to be returned on the date last stamped.